


MEB OYGM & Cisco işbirliği ile,

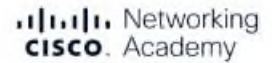
Dijital Transformasyon Python Programlama Eğitimleri

Kasım 2019



Enter the world of

<  python™



PCAP Programming Essentials in Python

Modul 2

1. Kısım

Hello World

- `print (“Hello, World!”)`
- İlk kodumuzun bileşenleri:
 - `print` sözcüğü
 - Parantez açma
 - Çift tırnak
 - Yazılacak metin: Hello, World!
 - Çift tırnak
 - Parantez kapama
- Her bir bileşenin yeri önemli

print() Fonksiyonu

- print sözcüğü bir fonksiyon adı belirtmektedir.
- Python fonksiyonları matematikteki fonksiyonlara göre daha çok içerik barındırır ve aynı zamanda esnekler.
- Bu bağlamda bir fonksiyon bir kod parçası içerisinde:
 - Bir etkiye sebep olabilir: Örneğin ekrana metin gönderme, dosya oluşturma vb.
 - Değer(sonuç) ya da değerler üretebilirler: Örneğin bir değerın karesi, bir metnin uzunluğu

print() Fonksiyonu

- Bu fonksiyonlar nereden gelmektedir:
 - Python içerisinde önceden tanımlı olabilirler. Örneğin print
 - Python eklentileri(ileride modül adını vereceğiz) ile gelebilirler.
 - Yazılımcı tarafından oluşturulmuş olabilirler.

print() Fonksiyonu

- Fonksiyon bileşenleri içerisinde çok önemli bir unsur da fonksiyonun argümanlarıdır.
- İhtiyaca göre bir fonksiyon argüman kullanabilir ya da kullanmayabilir.
- Bir fonksiyon tanımında muhakkak iki adet parantez bulunmalıdır. (açma ve kapama)
- İhtiyaç duyulan argümanlar bu parantezler arasına yerleştirilir.
- Not: Bir fonksiyon argümana ihtiyacı olmasa da parantezlere ihtiyaç duyar.

print() Fonksiyonu

- print fonksiyonuna verilen (fonksiyonda kullanılan) argüman ise bir metindir(karakter dizisi).
- Metin çift tırnak(ya da tek tırnak) arasına yazılmalıdır.
- Tırnaklar arasına yazılan metin yeni bir anlam kazanır ve kod olarak yorumlanmaz
- Tırnaklar arasına yazılan hemen hemen hersey (istisnalar var) veri olarak yorumlanır.
- Bir fonksiyonu çağırmak/kullanmak için fonksiyonun ismini ve varsa argümanları ile birlikte parantezleri kullanırız.

print() Fonksiyonu

- Python aşağıdaki gibi bir fonksiyon ile karşılaştığında *fonksiyon_ismi(argüman)*:
 - Öncelikle fonksiyon isminin geçerli olup olmadığına bakar. Eğer geçerli bir fonksiyon ismi değilse kod iptal edilir.
 - İkinci olarak kullanılan argümanların yeterli/doğru olup olmadığını kontrol eder. Eğer gereken sayıda argüman kullanılmamışsa kod çalışması iptal edilir.
 - Üçüncü olarak kullanılmak istenilen fonksiyonun tanımına bakarak argümanları bu fonksiyon tanımı içerisine aktarır.
- Fonksiyon çalıştırılır ve istenilen sonuç/etki elde edilir.

print() Fonksiyonu

- 2.1.1.7 LAB

Python

- Karmaşık bir program bir çok işlem/yönerge/talimat içerebilir.
- Python bu hususu düzenlemek için bir kurala ihtiyaç duyar:
 - Bir satır içerisinde sadece bir adet yönerge bulunmalıdır.-
- Bir satır boş olabilir(her hangi bir kod parçası içermeyebilir). Ama birden fazla yönerge içermemelidir.
- Not: Bu kuralı bozan bazı istisnalar vardır ilerde değinilecektir.

Python - \ -

- Metin içerisinde kullanılan ters kesme(bölüm) işaretinin - \ - özel bir anlamı vardır.
- Bu karaktere kaçış karakteri(kod dışı karakter) adı verilir.
- Diğer bir deyişle kaçış karakteri kendisinden sonra gelecek karakterin özel bir anlam içereceğini belirtir.
- Örneğin kaçış karakteri sonrası kullanılan n harfi yeni bir satıra(new line) geçilmesi gerektiğini belirtir.
- Kaçış karakteri ve n harfinden oluşan sembole yeni satır karakteri ismi verilir. - \n -

Python - \ -

- Kaçış karakteri kullanımında dikkat edilmesi gereken iki hususu vardır:
 - Eğer bir metinde ters kesme işareti kullanılacaksa, bu karakterin özel anlamını göz önünde bulundurarak, iki adet ters kesme işareti girilmelidir.
 - Ters kesme işareti ile kullanılan her karakter özel bir anlam içermeyebilir.

Python Fonksiyonları

- Aşağıda yazılı metne biraz yakından bakalım:
`print("Kara, kuru örümcek" , "hortuma" , "tırmandı.")`
- `print()` fonksiyonu 3 metin argüman ile birlikte çağırılmış.
- Her bir argüman arasına virgül konulmuş. Çift tırnak işareti ile virgül arasına boşluk koyulmuş fakat bu zorunlu bir durum değildir. Okumayı kolaylaştırmak amacıyla yapılmıştır.
- Çift tırnak içerisindeki virgül ile argümanları birbirinden ayırmamızı sağlayan virgül tamamen farklı roller üstlenmektedir

Python Fonksiyonları

- Aşağıda yazılı metne biraz yakından bakalım:
`print("Kara, kuru örümcek" , "hortuma" , "tırmandı.")`
- Metnin içerisinde kullanılan virgül konsol ekranında gözüktürken diğer virgül python yazım kurallarına uymak amacıyla kullanılmaktadır.
- Bu örnekten iki sonuç çıkartılabilir:
 - Bir print fonksiyonu içerisinde kullanılan birden fazla argüman tek bir satır içerisinde yazdırılır.
 - Python, print fonksiyonunun virgülle ayrılmış argümanlarını yazdırırken aralarına bir boşluk koyar.

Python Fonksiyonları

- Aşağıda yazılı metne biraz yakından bakalım:
`print("Kara, kuru örümcek" , "hortuma" , "tırmandı.")`
- Bu örneklerde print fonksiyonu içerisine yazılan argümanlar, konumları göz önünde bulundurularak ekrana çağrılmaktadır.
- Örneğin ikici sıraya yazılan argüman ikinci sırada çağrılmaktadır vb.

Python Fonksiyonları

- Python'da argümanlar farklı bir yöntem ile de kullanılabilir/çağrılabilir.
- Bu yöntem metnin içerisine yerleştirilen anahtar sözcükler ile argümanların kullanılması esasına dayanır.
- Bunun anlamı argümanların, artık sıra ile değil, argümanı belirtilen anahtar sözcüklere bakılarak çağrılmasıdır.

Python Fonksiyonları

- Burada iki özel anahtar sözcüğe de değinmek gerekmektedir.
- Bu anahtar sözcüklerin ilki “end” tir.
- “end” kullanırken dikkat edilmesi gereken iki husus:
 - Bir anahtar sözcük üç parçadan oluşur: Bunlar *anahtar sözcük*, *eşittir sembolü* ve *anahtar sözcüğün değeridir*.
Örn: end = “\t”
 - Bütün anahtar sözcükler konumları itibari ile metnin en sonunda yer almalıdır.
- end anahtar sözcüğünün varsayılan kullanımı end=“\n” şeklindedir.

Python Fonksiyonları

- 2.1.1.15 Örnek

Python Fonksiyonları

- 2.1.1.16 Örnek

Python Fonksiyonları

- Anahtar sözcük kullanarak print fonksiyonunun varsayılan, argümanlar arasına boşluk koyma, davranışı değiştirilebilir.
- ‘sep’ anahtar sözcüğü bu amaçla kullanılabilir.
‘sep’ = separator
- Bir fonksiyon çağrısında birden fazla anahtar sözcük kullanılabilir

Python Fonksiyonları

- 2.1.1.18 Örnek

Python Fonksiyonları

- 2.1.1.19 ve 2.1.1.20 sayfasındaki uygulamalar.

2. Kısım

Python Sabitleri (Literal)

- Sabitler yazılan programın içerisinde veri kodlamak ve bu verileri program içerisinde kullanmak amacıyla kullanılırlar.
- Python sabitlerinin değerleri, tanımlama esnasında belirlenir. İstenirse sonradan değiştirilebilir.

Örneğin: 123, 75, bir sabittir.

Python Sabitleri (Literal)

- Aşağıdaki örnekte iki tip sabit kullanıldığı görülmektedir:
 - İlk örnek bir metin/karakter-dizisi(string) tipindedir
 - İkinci örnek ise bir tamsayı(integer) tipindedir.

```
print("2")  
print(2)
```

Python Sabitleri (Literal)

- `print()` fonksiyonu her ne kadar konsolda her iki değişkeni aynı şekilde gösterse de bilgisayarın belleğinde bu iki değişken farklı şekillerde saklanmaktadır.
- Bir metin sadece yan yana dizilmiş karakter dizisini ifade eder.
- Sayılar ise bitlerden oluşan bir topluluktur.

Python Sabitleri (Literal)

- Sayılar bilgisayar ortamında (şimdilik) iki şekilde ifade edeceğiz:
Tamsayılar (Integers) ve Kesirli(Reel) sayılar(Floats)
- Bu iki tip birbirlerine ilk bakışta çok benzese de bilgisayar belleğinde nasıl tutuldukları ve kabul ettikleri değer aralıkları itibari ile farklılık göstermektedirler.
- Bir sayının değer aralığını, türünü ve uygulamasını belirleyen karakteristiğine tip(type) adı verilir.
- Python kodu içerisinde tanımlanan bir literal(sabit) ın tipi bellekte nasıl saklanacağını belirler.

Python Sabitleri (Literal)

- Python içerisinde kullanılan iki tipte daha gösterim vardır. Bunlardan ilki octal(sekizli-sekiz tabanında) gösterimdir.
- Herhangi bir tamsayı değişkeninin önüne 0O ya da 0o(sıfır ve o harfi) geldiğinde python bu değeri octal olarak yorumlar. Yani bu değer içerisindeki rakamlar [0-7] aralığındadır.
- İkinci gösterim ise hexadecimal(16'lı - 16 tabanlı) gösterimdir.
- Bu türden değişkenlerin önünde 0x ya da 0X(sıfır ve x) eki bulunur.

Python Sabitleri (Literal)

- Bilgisayarlar 2.5(iki buçuk) ya da -0.4(eksi sıfır nokta dört) gibi sayısal değerleri float olarak yorumlarlar.
- Bu değerler tanımlanırken nokta (.) kullanılır virgül (,) kullanılmaz.
- Python 4 ile 4.0 sayılarını tamamen farklı olarak algılar.
- 4 bir tamsayı (integer) tipinde iken 4.0 float tipinde algılanır.
- İkinci gösterilimde kullanılan nokta değişkenin float olarak algılanmasına neden olur.

Python Sabitleri (Literal)

- Bunun dışında matematiksel e (exponent - üst) ifadesi de bir değeri float haline getirir.
- Örneğin 3×10^8 : Üç kere on üzeri sekiz ifadesi Python da 3E8 ya da 3e8 şeklinde ifade edilir.
- E'nin bu gösterimdeki anlamı “10 üzeri” şeklindedir.
- Not: Bir üstel gösterimde üs değeri tam sayı olmalıdır.
- Not: Python bazı çıktıları E/e kullanarak gösterebilir.

Python Strings(Karakter Dizisi)

- Bir metin üzerinde işlem yapılmak istenildiğinde stringler (metin / karakter dizileri) kullanılır.
- String değişkenleri tanımlanırken tırnak ya da kesme işaretleri kullanılır.
- Eğer tanım bir tırnak ile başlamışsa tırnak ile, kesme işareti ile başlamışsa kesme işareti ile bitmelidir.
- Bir string değişkeninin içi boş olabilir.

Boolean

- Python içerisinde bir ifadenin doğru/yanlış olduğunu sorgulamak istediğimizde Boolean(Mantıksal) tipinde değişkenler kullanırız.
- Mantık cebrinde Doğru 1, Yanlış 0 ile ifade edilir.
- Bu iki değerın Python dilindeki karşılığı “True” ve “False” dur.
- Bu iki değişken büyük/küçük harf duyarlıdır. Tamamen görüldüğü şekilde yazılmalıdır.

Boolean

- 2.1.2.11 LAB

3. Kısım

Operatörler

- Programlama dillerinde operatörler, değerler üzerinde işlem yapmaya yarayan sembollerdir.
- *Veriler ve operatörler* birbirine bağlandığında *ifadeleri* oluştururlar. Sabit verinin(literal) kendisi en basit ifadedir.
- Operatörleri incelemeye aritmetik operatörlerle başlayacağız. (+, -, *, /, //, %, **)
- Aritmetikte olduğu gibi, + (artı) işareti iki sayı toplayabilen operatördür ve sonucu toplama'nın sonucuna eşittir.

Operatörler

- `**` (çift yıldız) işareti, üstel operatörüdür. Sol argümanı taban, sağ argümanı üs değeridir.
- Klasik matematik, (2^3 örneğinde olduğugibi) bu ifadeyi üst simgelerle yazmayı tercih eder.
- Saf metin editörleri bunu kabul etmez, bu yüzden Python da `**` kullanır ve bu ifade `2 ** 3` şeklinde yazılır.

Operatörler

- Üstel operatörün döndüreceği sonuç:
 - ** operatörünün iki argümanı da tam sayı ise sonuç tam sayı tipinde olacaktır.
 - ** operatörünün iki argümanından en az biri float(reel sayı) ise sonuç float (reel sayı) tipinde olacaktır.

Operatörler

- * (yıldız) işareti çarpma operatörüdür.
- / (kesme - eğik çizgi) işareti bölme operatördür.
- Eğik çizginin önündeki değer bölünen, eğik çizginin arkasındaki değer bölendir.
- Bölme operatörü tarafından üretilen sonucun tipi her zaman bir float(reel sayı)dır. Sonucun $1 / 2$ işlemindeki gibi bir reel sayı ya da $2 / 1$ işlemindeki gibi bir tam sayı görünümünde olması sonucun tipini değiştirmez.

Operatörler

- Eğer gerçekten bir tam sayı değeri sağlayan bir bölme işlemine ihtiyacınız varsa, Python'ın bunun için bir çözümü vardır.
- // (çift eğik çizgi) işareti bir tam sayı bölme operatörüdür. Standart bölme (/) operatöründen iki detayda farklıdır:
 - İşlem sonucunun kesirli kısmı yoktur. Ya kesirli kısım hiç bulunmaz (tam sayı için) ya da daima sıfıra eşittir (reel sayı için); yani sonuçlar her zaman yuvarlanır.
 - Bu kural hem tam sayı hem de reel sayı kullanıldığında geçerlidir.

Operatörler

- Tamsayı bölmesinin sonucu daima en yakın küçük tamsayıya yuvarlanır.
- Bu çok önemlidir: yuvarlamanın sonucu daima küçük tam sayıdır.
- Tam sayılı bölme aynı zamanda taban bölme olarak da adlandırılır.
- İleride bu terime tekrar değinilecektir.

Operatörler

- Python'da % (yüzde) işareti kalan operatördür.
- Operatörün sonucu, tam sayı bölmesinden sonra elde edilen kalandır.
- Başka programlama dillerinde operatöre modül operatörü de denir.
- Aşağıdakileri işlemler Python'ın hata üretmesine neden olur:
 - Sıfır ile yapılan bölme işlemi;
 - Sıfır ile yapılan tam sayı bölme işlemi;
 - Sıfır ile yapılan kalan/modül işlemi.

Operatörler

- Çıkarma operatörü - (eksi) işaretidir, ancak bu operatörün başka bir anlamı daha vardır.
- Bu operatörün diğer bir işlevi önüne geldiği sayının işaretini değiştirmektir.
- Çıkarma uygulamalarında eksi operatörü iki argüman bekler, argümanlar: sol (aritmetik terimle çıkartılan) ve sağ (aritmetik terimle çıkan).
- Diğer durumda ise tek bir argüman yeterlidir.

Operatörler

- Bazı operatörlerin diğerlerinden önce çalışmasına neden olan düzen, öncelikler hiyerarşisi olarak bilinir.
- Python, tüm operatörlerin önceliklerini tanımlar ve daha yüksek bir önceliğe sahip operatörlerin işlemlerinin düşük öncelikli operatörlerden önce yapılacağını varsayar.
- Operatörlerin yazılma sırası hesaplamaların sırasını ve sonucunu belirler.

Operatörler

- Bazı eşit önceliğe sahip operatörlerin yan yana yerleştirilmesinde hesaplama yine operatörlerin bağlanma sırasına bağlıdır.
- Python operatörlerinin çoğunda sola bağlanma vardır, yani ifadenin hesaplanması soldan sağa yapılır.
- Bu basit örnek ile sola bağlanmanın nasıl çalıştığını gözlemleyebiliriz:

print (9 % 6 % 2)

soldan sağa: ilk önce $9 \% 6$, 3 verir; sonra $3 \% 2$, 1 verir

Operatörler

- ÖNEMLİ
- Üstel operatörü sağa bağlanma kullanır.
- Örneğin:

```
print (2 ** 2 ** 3)
```

$2 ** 3 \rightarrow 8$; $2 ** 8 \rightarrow 256$

Operatörler

- Aritmetik işlemlerde olduğu gibi, bir ifadenin hesaplanmasında öncelikle parantez içindeki kısımlar hesaplanır.

Priority	Operator	
1	<div><div>+</div><div>-</div></div>	unary
2	<div><div>*</div><div>*</div></div>	
3	<div><div>*</div><div>/</div><div>%</div></div>	
4	<div><div>+</div><div>-</div></div>	binary

4. Kısım

Değişkenler

- Değişkenler veri depolamak için kullanılan özel alanlardır.
- Değişkenler otomatik olarak bir programda görünmez. Geliştirici olarak siz, programlarınızda kaç tane ve hangi değişkenleri kullanacağınıza karar vermelisiniz.
- Her Python değişkeni şunları içerir:
 - bir isim,
 - bir değer (depo alanının içeriği)

Değişkenler

- Bir değişkene isim verirken dikkat edilmesi gerekenler:
 - Sadece büyük harf, küçük harf, rakam ve _ (alt çizgi) karakterleri kullanılmalıdır.
 - Değişkenin adı bir harfle başlamalıdır.
 - Alt çizgi karakteri(_) bir harf olarak değerlendirilebilir.
- Büyük ve küçük harfler farklı olarak ele alınır (Alice ve ALICE aynı isimdir, ancak Python bunu farklı değişken olarak yorumlar. Dolayısıyla iki farklı değişkendir)
- Değişken ismi Python'da kullanılan (özel olarak ayrılmış) anahtar kelimeler olmamalıdır.

Değişkenler

- Python, değişken isimlerinin uzunluğu konusunda kısıtlamalar getirmez.
- Ancak bu durum uzun değişken isimlerinin, kısa değişken isimlerinden daha iyi olduğu anlamına gelmez.
- Python yalnızca Latin harfleri değil, diğer alfabelere özgü karakterleri de kullanmanıza izin verir.

Değişkenler

- PEP 8 - (Python Kodu için Stil Kılavuzu) Python dilinde değişken ve metot/fonksiyon isimlerinin kullanımını için aşağıdaki şartları önerir:
 - 1) Değişken isimleri, okunabilirliği artırmak için, küçük harflerden ve gerekiyorsa alt çizgi ile ayrılmış kelimelerden oluşmalıdır. (Örn: degisken, benim_degiskenim).
 - 2) Metot isimleri değişken isimleriyle aynı kuralı izler (Örn: metod, benim_metodum)

Değişkenler

- PEP 8 – (Python Kodu için Stil Kılavuzu) Python dilinde değişken ve metot/fonksiyon isimlerinin kullanımı için aşağıdaki şartları önerir:
3) Büyük harf ve küçük harfleri beraber kullanmak da mümkündür (örneğin, benimDegiskenim). Yalnızca daha önce kullanılmış olan yapının devamlılığı, kullanım için daha uygundur.

Değişkenler

- Aynı kısıtlamaların metod adları için de geçerli olduğunu unutmayalım.
- Python'da anahtar kelimeler veya daha doğrusu ayrılmış anahtar kelimeler vardır.
- Bunların özel bir anlam içerdikleri için isimlendirmede kullanılmaması gerekir.
- Ne değişkenler ne fonksiyonlar ne de oluşturmak istenilen diğer tipler için isim olarak kullanılmamalıdır.

Değişkenler

- ['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
- Ayrılmış kelimenin anlamı önceden tanımlanmıştır ve hiçbir şekilde değiştirilmemelidir.

Değişkenler

- Şu ana kadar öğrendiğimiz ve ileride öğreneceğimiz veri tiplerini kullanarak bir değer saklamak için değişkenler kullanılır.
- Bir değişken ona bir değer atanmasıyla var olur.
- Diğer dillerden farklı olarak, Python dilinde bir değişkenin tipinin deklare edilmesine(belirtilmesine) gerek yoktur.
- Var olmayan bir değişkene herhangi bir değer atarsanız, değişken otomatik olarak oluşturulur.

Değişkenler

- Oluşturma (ya da başka anlatım ile - sözdizimi) son derece basittir: sadece istediğiniz değişkenin adını, ardından eşittir işaretini (=) ve değişkene eklemek istediğiniz değeri kullanın.
Örneğin: `degisken = 1`
- İlk bölüm `degisken` adlı bir değişken oluşturur ve 1'e eşit bir tamsayı değerine sahip bir sabit veri atar.

Değişkenler

- Örnek:
print () ifadesini kullanarak söz dizisi ve değişken çıktısı yazdırabilmek için, metin ve değişkeni + operatörü ile birleştirebilirsiniz.
var = "3.7.1"
print("Python version: " + var)

Değişkenler

- Eşittir işareti aslında bir atama operatörüdür. Bu garip gelse de, operatör basit bir sözdizimine ve basit bir açıklamaya sahiptir.
- Sağ argümanının değerini sola atar, sağ argüman ise sabit verileri, operatörleri ve önceden tanımlanmış değişkenleri içeren karmaşık bir ifade olabilir.

Değişkenler

- Aşağıdaki koda bakın:

benim_sayım = benim_sayım + 1

- Bu satır, değişkeninin değerini 1 ile toplayıp kendisine yeni değer olarak atıyor.
- Böyle bir kaydı görünce, bir matematikçi muhtemelen protesto ederdi - hiçbir sayı kendisi artı bir sayıya eşit olamaz. Bu bir çelişkidir. Ancak Python, = işaretine, matematiksel eşit değil, bir değer atama işlemi gibi davranır.

Değişkenler

- Aşağıdaki koda bakın:

benim_sayım = benim_sayım + 1

- Peki programda böyle bir kaydı nasıl okursunuz?
benim_sayım değişkeninin geçerli değerini alın, buna 1 ekleyin ve sonucu benim_sayım değişkeninde saklayın.

Değişkenler

- 2.1.4.6 ÖRNEK

Değişkenler

- 2.1.4.7 LAB

Değişkenler

- $x = x * 2$ işlemi ile $x *= 2$ işlemi aynıdır.
- Operatör, iki değişkenli bir operatör ise (bu çok önemli bir koşuldur) ve operatör aşağıdaki bağlamda kullanılmış ise:
değişken = değişken *Operatör* ifade
- Aşağıdaki gibi basitleştirilebilir:
değişken *Operatör* = ifade

Değişkenler

- *2.1.4.9 LAB*
- *2.1.4.10 LAB*

5. Kısım

Yorum Ekleme

- Programa eklenmiş bir açıklama, çalışma zamanında çıkarılır ve bu yorum olarak adlandırılır.
- Python, bir yorumla karşılaştığında, yorum Python için tamamen şeffaftır - Python'un bakış açısından bu sadece bir boşluktur.
- Python'da, yorum # işaretiyle başlayan ve satırın sonuna kadar devam eden bir metindir.

Yorum Ekleme

- Yorum yazımıyla birlikte değişken isimlerini kendini açıklayacak şekilde kullanmak en iyi yöntemdir.
- Birden fazla kod satırını hızlıca yoruma almak veya birden çok satırı toplu olarak yorum yapmak istiyorsanız, değiştirmek istediğiniz satırları seçin ve aşağıdaki klavye kısayolunu kullanın:
- CTRL + / (Windows) veya CMD + / (Mac OS).

Yorum Ekleme

- 2.1.5.2 LAB

Yorum Ekleme

- Birkaç satırdan oluşan bir yorum yapmak istiyorsanız, hepsinin önüne # işareti koymanız gerekir. Ayrıca, o anda gerekmeyen bir kod parçası için de yorum işaretini kullanabilirsiniz.
- Bir süre sonra kendi kodunuzu okurken veya başkaları kodunuzu okurken yorumlar önemli olabilir.

6. Kısım

input() Fonksiyonu

- input () fonksiyonu kullanıcı tarafından girilen verileri okur ve aynı verileri çalışan programa gönderir.
- Program, kullanıcıdan konsoldan veri girmesini ister. (muhtemelen klavyeyi kullanarak, ancak ses veya görüntü kullanarak veri girişi yapmak da mümkündür).

```
X = input("Bir metin giriniz: ")
```

input() Fonksiyonu

- input() fonksiyonu, içerisine değişken almadan çağırılabilir.
- input() fonksiyonu konsolu veri giriş moduna geçirecektir.
- Bu fonksiyon çalıştırıldığında konsolda yanıp sönen bir imleç meydana gelir.
- Veririn girilmesinin ardından enter tuşuna basarak veri bitirilir ve bu moddan çıkılır.
- Girilen tüm veriler fonksiyonun sonucu olarak programınıza gönderilecektir.
- Sonucun bir değişkene atanması gerekir. Bu adımı atlamak, girilen verilerin kaybolmasına neden olur.

input() Fonksiyonu

- input () fonksiyonu, print () fonksiyonunu kullanmadan kullanıcıya çıktı üretebilir.

```
girdi = input ("Bana bir şey söyle ...")
```

```
print ("Hmm ...", girdi, "... Gerçekten mi?")
```

- input () fonksiyonu bir argümanla çağırılmıştır. Bu argümanın tipi bir stringtir. (karakter dizisi)
- Kullanıcıya bir mesaj girmeden argüman konsolda gösterilir; input() daha sonra işini yapar.
- input() fonksiyonunun sonucunun tipi de bir stringdir.

Tip Dönüşümü

- Python veri tiplerini dönüştürmek için bazı hazır fonksiyonlar sunar:
`int()` ve `float()`.
- `int()` fonksiyonu bir argüman alır ve argümanı bir tam sayıya dönüştürür.
Örneğin: `int(string)`, `int(float)`
- `float()` fonksiyonu bir argüman alır ve argümanı float (reel sayı) tipine dönüştürür.
Örneğin: `float(string)`

Tip Dönüşümü

- 2.1.6.4 Örnek
- 2.1.6.5 Örnek

Tip Dönüşümü

- + (artı) işareti iki string tipindeki değişkene uygulandığında bir birleştirme operatörü olur:

string + string

- Bu durumda + basitçe iki stringi birleştirir (yapıştırır). Sol taraftan başlayarak birleştirme yapılır.
- Aritmetik operatör olarak kullanımının aksine, + birleştirme operatörü olarak kullanıldığında sıra önemlidir. Yani *"ab" + "ba"* ile *"ba" + "ab"* aynı değildir.

Tip Dönüşümü

- * (yıldız) işareti, bir dizeye tam sayıyla uygulandığında (sıra önemsiz olarak kullanılabilir) kopyalama metodu olur:
*dizi * tam_sayı* veya *tam_sayı * dizi*
- 2.1.6.7 Örnek

Tip Dönüşümü

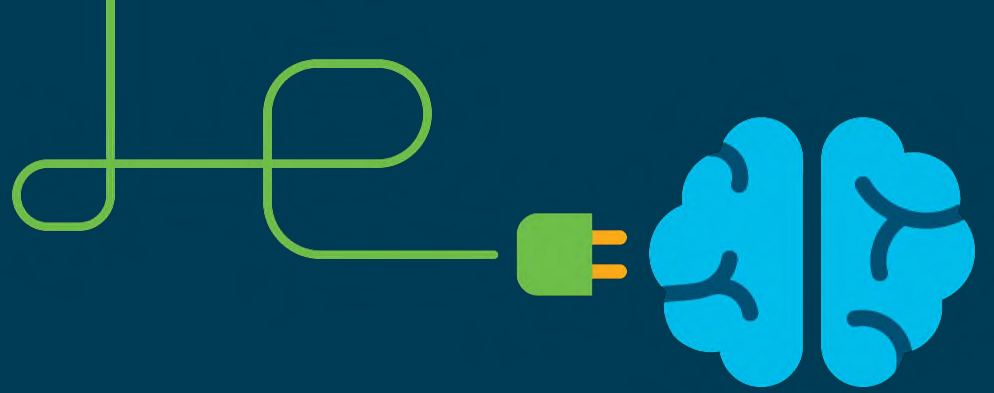
- `str()` metodu tam sayıları dizelere çevirmek için kullanılabilir.

`str(tam_sayı)`

Tip Dönüşümü

- 2.1.6.9
- 2.1.6.10
- 2.1.6.11






MEB OYGM & Cisco işbirliği ile,

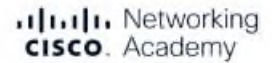
Dijital Transformasyon Python Programlama Eğitimleri

Kasım 2019



Enter the world of

<  python™



PCAP Programming Essentials in Python

Modul 3

1. Kısım

Python

- Yazılım geliştiriciler bir program yazar ve bu program bazı sorular ve hesaplar barındırır.
- Bilgisayarlar bu programı çalıştırır ve bu sorulara bazı yanıtlar sunar. Program bu yanıtlara uygun davranış sergileyebilmelidir.
- Python soru sorulmasına imkan sağlayan bazı özel operatörler kullanır.
- Bilgisayarlar kendisine yöneltilen sorulara iki türlü cevap verirler: Evet, doğru ve Hayır, yanlış

Python (==)

- Eşitlik operatörü (==) sağındaki ve solundaki değerlerin eşit olup olmadığını kıyaslar.
- Eğer eşitse sonuç True , (doğru)
- Eşit değilse sonuç False , (yanlış) çıkar.

Python (!=)

- Eşitsizlik operatörü (!=) de sağındaki ve solundaki değerlerin eşit olup olmadığını kıyaslar.
- Eğer eşitse sonuç False , (yanlış)
- Eşit değilse sonuç True , (doğru) çıkar.

Python (>, >=)

- Büyüktür operatörü de (>) solunda girili değerin, sağındaki değerden büyük olup olmadığını kıyaslar.
- Eğer büyükse sonuç True (doğru), büyük değilse sonuç False (yanlış) çıkar.
- Büyüktür operatörünün farklı bir türü büyük eşittir (>=) operatörüdür.
- Eğer sol taraf sağ taraftan büyük ya da sağ tarafa eşitse sonuç True (doğru), küçükse sonuç False (yanlış) çıkar.

Python (<, <=)

- Küçüktür operatörü (<)ve küçük eşittir operatörü (<=)de benzer kıyaslar yapmamızı sağlar.
- Küçüktür operatörü kullanıldığında sol taraf sağ taraftan küçükse sonuç True (doğru), değilse sonuç False (yanlış) çıkar.
- Küçük eşittir operatörü kullanıldığında sol taraf, sağ taraftan küçük ya da sağ tarafa eşitse sonuç True (doğru), büyükse sonuç False (yanlış) çıkar.

Python Operatör Önceliği

Priority	Operator	
1	<code>+</code> , <code>-</code>	unary
2	<code>**</code>	
3	<code>*</code> , <code>/</code> , <code>//</code> , <code>%</code>	
4	<code>+</code> , <code>-</code>	binary
5	<code><</code> , <code><=</code> , <code>></code> , <code>>=</code>	
6	<code>==</code> , <code>!=</code>	

Python Operatör Önceliği

- Örnek 3.1.1.5

Python Koşullu Durumlar

- Kod içerisinde, koşula bağlı durumlar kullanılması gerekiyorsa Python içerisinde tanımlı bazı ifadeler kullanılabilir.
- Bu ifadelerden ilki “*if*” tir. Bir “*if*” ifadesinin en basit şekli aşağıdaki gibidir.

if doğru_veya_yanlış:

doğru_ise_buradaki_işlemleri_yap

Python Koşullu Durumlar

- Bu ifadeyi yazarken dikkat edilmesi gereken hususlar aşağıdaki şekilde sıralanabilir:
 - *if* Anahtar sözcüğü
 - En az bir, ya da daha fazla boşluk,
 - Mantıksal bir ifade (yalnızca True ya da False değeri alabilecek)
 - İki nokta ve yeni satıra geçiş.
 - *if* bloğu içerisindeki ifadeler. (Indentation kuralına dikkat edilmelidir.)

Python Koşullu Durumlar

- Indentation(Girinti) kuralı iki şekilde sağlanabilir. “*if*” bloğu içerisinde tab kullanarak ya da belirli sayıda boşluk (tavsiye edilen 4 adet) bırakarak.
- Blok içerisindeki satırların hepsi aynı sayıda boşluk içermelidir.

if true_or_not:

do_this_if_true

do_this_if_true

Python Koşullu Durumlar

- Bir *if* bloğu aşağıdaki şekilde çalışır.
 - Eğer `true_or_not` ile belirtilen mantıksal ifade `True` ise indentation ile belirtilen kod bloğu çalıştırılır.
 - Eğer `true_or_not` ile belirtilen mantıksal ifade `False` ise indentation ile belirtilen kod bloğuna bakılmadan bir sonraki satırdan çalışmaya devam edilir.

if true_or_not:
 do_this_if_true

Python if/else

- Eğer koşul şartı sağlanmamışsa, yapılması gerekenleri bir “*else*” bloğu ile ifade edebiliriz. Bu şekilde tanımlanan ifadelere if/else ifadesi denir.

if true_or_false_condition:

perform_this_if_condition_true

else:

perform_this_if_condition_false

Python if/else

- Else bloğu içerisinde tanımlanan ifade, if koşulunun sağlanmadığı durumlarda ne yapılacağını belirtir.
- Bir if/else bloğu aşağıdaki şekilde çalışır:
 - Eğer `true_or_false_condition` ile belirtilen mantıksal ifade True ise indentation ile belirtilen kod bloğu çalıştırılır. “else” bloğu çalıştırılmaz.
 - Eğer `true_or_false_condition` ile belirtilen mantıksal ifade False ise “if” bloğu çalıştırılmaz ve “else” bloğu çalıştırılır.

Python Nested-if

- Eğer birden fazla koşul durumu kontrol edilmesi gerekiyorsa Python bunun için farklı çözümler sunabilir.
- Bunlardan ilki iç içe yerleştirilmiş if ifadeleridir. (Nested-if)

Python Nested-if

```
if hava_güzelse:  
    if güzel_bir_restoran_bulabildiysen:  
        restoranda_yemeğini_ye()  
    else:  
        sandviç_ye()  
else:  
    if bilet_bulabildiysen:  
        tiyatroya_git()  
    else:  
        alışveriş_yap()
```

Python if/elif

- İkinci bir çözüm ise “*elif*” anahtar sözcüğünün kullanılmasıdır. “*elif*” sözcüğü else-if sözcüklerinin birleşiminden oluşur.
- Bu tür kullanımlara ardışık(kaskad) if ifadesi adı verilir.

Python if/elif

```
if hava_güzelse:  
    yürüyüş_yap()  
elif bilet_bulabildiysen:  
    tiyatroya_git()  
elif sevdiğin_restoran_açıkça:  
    yemeğini_ye()  
else:  
    evde_satranç_oyna()
```

Python if/elif

- Dikkat edilmesi gereken bir kaç nokta:
 - *if* kullanmadan *else* kullanılamaz.
 - İster *elif* kullanılsın ister kullanılsın, *else* ifadesi daima koşulların sonunda kullanılır.
 - *else* ifadesinin kullanılması zorunlu değildir.
 - Ardışık *if* ifadelerinde *else* kullanılmışsa en azından bir ifade bloğu çalıştırılacaktır.
 - Ardışık *if* ifadelerinde *else* kullanılmamışsa hiç bir ifade bloğu çalıştırılmayabilir.

Python if/elif

```
number1 = int(input("Enter the first number: "))  
number2 = int(input("Enter the second number: "))
```

```
if number1 > number2:  
    larger_number = number1  
else:  
    larger_number = number2
```

```
print("The larger number is:", larger_number)
```

Python if/elif

```
number1 = int(input("Enter the first number: "))  
number2 = int(input("Enter the second number: "))
```

```
if number1 > number2: larger_number = number1  
else: larger_number = number2
```

```
print("The larger number is:", larger_number)
```

Python if/elif

- 3.1.1.9 Örnek 3

Python if/elif

- Kodun bir parçasının birden fazla çalıştırılmasına döngü(loop) adı verilir.
- 2. ile 8. satır arası kodlar bir döngü oluşturmaktadır.

```
satır 01 en_büyük_sayı = -999999999  
satır 02 sayı = int(input("Bir sayı giriniz: "))  
satır 03 if sayı == -1:  
satır 04     print("Bir şey söyle")  
satır 05     koddan_çık()  
satır 06 if sayı > en_büyük_sayı:  
satır 07     en_büyük_sayı = sayı  
satır 08 02 ile başlayan satırdan devam et
```

Python if/elif

- 3.1.1.11 LAB
- 3.1.1.12 LAB
- 3.1.1.13 LAB

2. Kısım

Python - Döngüler

- Python içerisinde bir döngü “while” ile de tanımlanabilir.
while koşul_ifadesi:
döngü_gövdesi
- Eğer *koşul_ifadesi*’nin sonucu False olarak hesaplarsa *döngü_gövdesi* yürütülmez/çalıştırılmaz.
- Burada dikkat edilmesi gereken en önemli husus bir şekilde *koşul_ifadesi*’nin değişebilmesidir. Eğer *koşul_ifadesi* daima True olarak hesaplanırsa sonsuz bir döngü içerisine girilmiş olur.

Python - Döngüler

```
en_büyük_sayı = -999999999
sayı = int(input("Bir sayı girin ya da durmak için -1 yazın: "))

while sayı != -1:
    if sayı > en_büyük_sayı:
        en_büyük_sayı = sayı
    sayı = int(input("Bir sayı girin ya da durmak için -1 yazın: "))
print("En büyük sayı: ", en_büyük_sayı)
```

Python - Döngüler

- Aşağıdaki kodu inceleyelim.

```
sayac = 5  
while sayac != 0:  
    print("Döngü içerisindeyim.", sayac)  
    sayac -= 1  
print("Döngü dışındayım", sayac)
```

Python - Döngüler

- Bu kod sayac değeri sıfıra ulaşıncaya kadar “Döngü içerisindeyim” ve sayacın o anki değerini ekrana yazdıracaktır.
- Sayaç değeri sıfır olduğunda ise “Döngü dışındayım” ve sayacın değerini yazdıracaktır.

```
while sayac != 0:
```

```
    print(“Döngü içerisindeyim. ”, sayac)
```

```
    sayac -= 1
```

```
print(“Döngü dışındayım”, sayac)
```

Python - Döngüler

- 3.1.2.3 LAB

Python - Döngüler

- Python içerisinde yazılabilecek bir başka döngü ise “*for*” döngüsüdür.
- “*for*” döngüleri, bir döngüyü belli bir sayıda çalıştırmak istenildiği durumlarda kullanılabilir.
- “*for*” döngüleri çok büyük veri kümeleri üzerinde çalıştırılabilir. Örn:

```
for i in range(100):  
    #birşeyler_yap()
```

Python - Döngüler

- Bir for döngüsün yazılma adımları:
 - 1) “*for*” anahtar sözcüğü ile başlanır.
 - 2) “*for*” dan sonra gelen ifade kontrol değişkeni olarak adlandırılır. (*i*)
 - 3) “*in*” anahtar sözcüğü kontrol değişkenine atanacak muhtemel değerleri tarif etmek için kullanılır.
 - 4) *range()* fonksiyonu, kontrol değişkenine atanacak bütün değerlerin kontrolünden sorumludur. 0 ile 99 arasındaki bütün tamsayı değerlerini sıra ile *i* kontrol değişkenine atar.

Python - Döngüler

- *range()* fonksiyonu tek bir argüman ile çağrıldığında sıfırdan başlar ve girilen değerden küçük en büyük tamsayıya kadar, birer birer artarak devam eder.
- *range()* fonksiyonu iki argüman ile de çağrılabilir. Böyle bir durumda girilen ilk değer başlangıç değerini, ikinci değer ise kontrol değişkeninin atanmayacağı ilk değeri gösterir.

Örn:

```
for i in range(2, 8):  
    print("i'nin şimdiki değeri:", i)
```


Python - Döngüler

- *range()* fonksiyonu üç argüman ile de çağrılabilir. Bu durumda üçüncü argüman kontrol değişkeninin her döngü içerisinde ne kadar arttırılacağını belirtir.

```
for i in range(2, 8, 2):
```

```
    print("i'nin şimdiki değeri:", i)
```

- Not: *range()* fonksiyonunun ikinci değeri, mutlaka birinci değerinden büyük olmalıdır.

Python - Döngüler

- 3.1.2.6 LAB

Python - Döngüler

- Bazı durumlarda döngünün çalıştırılması işlemine bir son verilmek istenebilir. Böyle bir durumda döngü gövdesine “*break*” anahtar sözcüğü yazılmalıdır.
- “*break*” anahtar sözcüğü kullanıldığında döngüden çıkılır ve döngü işlemine son verilir. Sıradaki satırların çalıştırılmasına devam edilir.

Python - Döngüler

- Bazen de döngü gövdesinde yapılan işlemleri bir seferlik atlamak gerekebilir. Böyle bir durumda döngü gövdesine “*continue*” anahtar sözcüğü yazılmalıdır.
- “*continue*” anahtar sözcüğü, sanki döngü gövdesinde yazılan işlemler yapılmışçasına, sıradaki kontrol değerine geçilmesini sağlar.

Python - Döngüler

- 3.1.2.8 ÖRNEK
- 3.1.2.9 LAB
- 3.1.2.10 LAB
- 3.1.2.11 LAB

Python - Döngüler

- Döngülerin içerisinde, “if” koşullu durumlarında olduğu gibi, “else” ifadeleri kullanılabilir.

```
i = 1  
while i < 5:  
    print(i)  
    i += 1  
else:  
    print("else:", i)
```

Python - Döngüler

```
i = 111
```

```
for i in range(2, 1):
```

```
    print(i)
```

```
else:
```

```
    print("else:", i)
```

- Eğer herhangi bir nedenle döngü gövdesi çalıştırılmazsa, kontrol değişkeni döngüden önceki değerini korur.
- Eğer kontrol değişkeni for döngüsünden önce oluşturulmamışsa, else ifadesi içerisinde oluşturulmaz.

Python - Döngüler

- 3.1.2.14 LAB
- 3.1.2.15 LAB

3. Kısım

Mantıksal İşlemler

- Python da mantıksal birleşim ve mantıksal ayrışım işlemleri için “*and*”(ve) ve “*or*”(veya) operatörleri kullanılır.
- Bu operatörler binary operatörlerdir. Bit’ler üzerinde çalışırlar.
- İşlem öncelikleri kıyaslama operatörlerinden düşüktür.
- Bu operatörler ile yapılan işlemlerin sonucu doğruluk tablosuna dayanır.

Mantıksal İşlemler

Argument A	Argument B	A and B
False	False	False
False	True	False
True	False	False
True	True	True

Mantıksal İşlemler

Argument A	Argument B	A or B
False	False	False
False	True	True
True	False	True
True	True	True

Mantıksal İşlemler

- “*or*” operatörünün öncelik sırası “*and*” den sonra gelir.
- Bir diğer mantıksal operatör “*not*”(değil) operatörüdür. True değerini False, False değerini True yapar.
- İşlem önceliği + ve - ile aynıdır.

Mantıksal İşlemler

Argument	not Argument
False	True
True	False

Mantıksal İşlemler

- De Morgan Kuralı:

$$\text{not } (p \text{ and } q) == (\text{not } p) \text{ or } (\text{not } q)$$

$$\text{not } (p \text{ or } q) == (\text{not } p) \text{ and } (\text{not } q)$$

- Mantıksal operatörleri sağ ve solundaki değerleri(kaç bitten oluşursa oluşsun) bir bütün olarak ele alırlar.
- Bu operatörler sadece sıfır değerini (bütün bitlerin sıfır olması) *False*; sıfır olmama değerini (en az bir bitin bir olması) *True* olarak yorumlar.

Mantıksal İşlemler

- Bitler üzerinde işlem yapmamızı sağlayan 4 operatör vardır. Bunlara bitwise (bitisel) operatörler denir. Bunlar:

& (ampersand) - bitisel birleşim;

/ (bar) - bitisel ayrışım;

~ (tilde) - bitisel olumsuzluk;

^ (caret) - bitisel ayrıcalıklı veya (xor).

Mantıksal İşlemler

- Bitsel operatörlerin argümanları tamsayı(integer) olmalıdır. Float değerler kullanılmamalıdır.
- Mantıksal operatörler ile bitsel operatörler arasındaki temel fark; mantıksal operatörler bit seviyesinde işlem yapmazlar.
- Bitsel operatörler ise bit bit işlem yaparlar.
 - & (ampersand) - bitsel birleşim;
 - / (bar) - bitsel ayrışım;
 - ~ (tilde) - bitsel olumsuzluk;
 - ^ (caret) - bitsel ayrıcalıklı veya (xor).

Mantıksal İşlemler

- 3.1.3.3 Örnek

Mantıksal İşlemler

- Python’da bitler üzerinde işlem yapmamızı sağlayan bir başka operatör de bulunmaktadır. Bu operatörün ismi *shifting* (kaydırma) operatörüdür.
- Bu operatör de sadece tamsayılar ile çalışmaktadır.
- Onluk tabanda bulunan çarpma ve bölme işlemine benzemektedir. Fakat ikili tabanda çalışmaktadır.
- Bir değeri bir bit sola kaydırmak onu iki ile çarpmaya karşılık gelmektedir.
- Bir değeri bir bit sağa kaydırmak onu ikiye bölme anlamı taşımaktadır.

Mantıksal İşlemler

Sola kaydırma: *değer << bit_adedi*

Sağa kaydırma: *değer >> bit_adedi*

- Operatörün solunda kalan kısım bit kaydırma işlemi yapılacak değeri belirtirken, operatörün sağında yer alan kısım kaç bit kaydırmanın yapılacağını belirtir.

Mantıksal İşlemler

Priority	Operator	
1	<code>~</code> , <code>+</code> , <code>-</code>	unary
2	<code>**</code>	
3	<code>*</code> , <code>/</code> , <code>//</code> , <code>%</code>	
4	<code>+</code> , <code>-</code>	binary
5	<code><<</code> , <code>>></code>	
6	<code><</code> , <code><=</code> , <code>></code> , <code>>=</code>	
7	<code>==</code> , <code>!=</code>	
8	<code>&</code>	
9	<code> </code>	
10	<code>=</code> , <code>+=</code> , <code>-=</code> , <code>*=</code> , <code>/=</code> , <code>%=</code> , <code>&=</code> , <code>^=</code> , <code> =</code> , <code>>>=</code> , <code><<=</code>	

4. Kısım

Listeler

- Şimdiye kadar, oluşturduğumuz değişkenlere sadece bir değer depolamayı öğrendik.
- Bu değişkenlere matematik analogisinde skalerler denir.
- Şimdiye kadar kullandığınız tüm değişkenler aslında skalerdir.
- Bir değişken için birden fazla değer depolamak kullanışlı olabilir.

Listeler

- Böyle çok değerli bir değişken nasıl oluşturulur?
- Listeler sağa bakan köşeli parantez ile başlar ve sola bakan köşeli parantez ile sona erer;
- Bir listenin içindeki elemanların tipleri farklı olabilir.
- Python, bir listedeki öğelerin daima sıfırdan başlayarak numaralandırıp saklar.
- Liste bir elemanlar topluluğudur, fakat her eleman bir skalerdir.

Listeler

- Listeden bir eleman seçmeyi sağlayan köşeli parantez içindeki değere **indeks** denir.
- Listedен bir eleman seçme işlemi **indeksleme** olarak bilinir.
- Şimdiye kadar kullandığımız tüm değişken verileri sabit veriydi(literal). Değerleri çalışma zamanında değişmezdi. Fakat herhangi bir ifade de indeks olabilir.

Listeler

- Örnek 3.1.4.2

Listeler

- Bir listenin uzunluğu programın çalışması sırasında değişebilir.
- Listenin geçerli uzunluğunu kontrol etmek istiyorsanız, *len()* adındaki bir fonksiyon kullanılır.
- Fonksiyon, listenin ismini bir argüman olarak alır ve o anda listede bulunan öğelerin sayısını döndürür. (başka bir deyişle - listenin uzunluğu).

Listeler

- Listedeki öğelerin herhangi biri herhangi bir zamanda silinebilir – bu işlem, *del()* adlı bir talimatı ile yapılır. Not: Bu bir talimattır, fonksiyon değildir.
- Silme işlemi için silinecek elamanın işaret edilmesi gerekir.
- Bu eleman listeden yok olacak ve listenin uzunluğu da bir azalacaktır.
- Var olmayan bir liste elemanına erişemezsiniz
- Olmayan bir liste indeksinin hem değerini alınamaz hem de olmayan indekse bir değer atayamazsınız.

Listeler

- Örnek 3.1.4.4

Listeler

- Negatif tam sayı olarak verilen indeksler Python'da kullanılabilir ve çok faydalı olabilir.
- İndeksi -1'e eşit olan bir eleman listedeki son elemandır.
- Benzer şekilde, indeksi -2 olan eleman ise, listedeki sondan bir önceki elemandır.

Listeler

- 3.1.4.6 LAB

Fonksiyon ve Metot

- Metotlar fonksiyonların özel bir türüdür.
- Bir fonksiyon gibi davranır ve bir fonksiyon gibi görünür, fakat nasıl çalışacağı ve çağırılma tarzları farklıdır.
- Bir fonksiyon hiçbir veriye ait değildir - veri alır, yeni veri yaratabilir ve (genellikle) bir sonuç üretir.
- Metotlar tüm bunları yapar, Bununla birlikte seçilen bir varlığın durumunu da değiştirebilir.

Fonksiyon ve Metot

- Bir metot, fonksiyonların aksine, bir veriye aittir. Bu data için çalışır. Bir Fonksiyon ise bütün koda aittir.
- Bu özellik metodun çağırılması için, metodun çağırıldığı verinin bazı özelliklerinin olması gerektiği anlamına gelir.

Fonksiyon ve Metot

- Genel olarak, tipik bir fonksiyon çağrısı şöyle görünebilir:
sonuc = function(arg)
- Fonksiyon bir argüman alır, bir şey yapar ve bir sonuç verir.
- Tipik bir metot çalıştırma ise genellikle şöyle görünür:

sonuc = data.method(arg)

- Metot adından önce, metoda sahip olan verinin adı gelir. Ardından bir nokta, metot adı ve parantez içerisinde argümanlar eklenir.
- Metot bir fonksiyon gibi davranabilir, ancak daha fazlasını da yapabilir, çağrıldığı verinin durumunu değiştirebilir.

Fonksiyon ve Metot

list.append(argüman)

- Bu metot, argümanın değerini alır ve metodun sahibi olan listenin sonuna yerleştirir.
- Listenin uzunluğu bir artar.

Fonksiyon ve Metot

list.insert(yer, argüman)

- insert () yöntemi biraz daha yeteneklidir.
 - Yalnızca liste sonuna değil listedeki herhangi bir yere yeni bir eleman ekleyebilir.
 - İki argüman alır:
 - Birincisi, eklenecek elemanın istenen yerini belirtir;
- Not: Yeni elemanın sağında yer alan bütün elemanlar (istenen pozisyonundaki dahil) yeni elemana yer açmak amacıyla sağa kaydırılır;
- İkincisi, eklenecek öğedir.

Fonksiyon ve Metot

- Bir liste oluşturulduğunda boş olarak tanımlanabilir. (bu işlem bir köşeli parantez çifti ile yapılır) ve ardından bir döngü ile yeni elemanları bu boş listeye ekleyebilirsiniz.

```
myList = [] # boş bir liste oluşturuldu  
for i in range(5):  
    myList.insert(0, i + 1)  
print(myList)
```

Fonksiyon ve Metot

```
myList = [10, 1, 8, 3, 5]
```

```
total = 0
```

```
for i in myList:
```

```
    total += i
```

```
print(total)
```

Fonksiyon ve Metot

- Bir listenin elemanlarını yeniden düzenlemek gerekirse, mesela elemanların sırasını tersine çevirmek
- Python, veri değişmek için pratik bir yol sunar:

degisken1 = 1

degisken2 = 2

degisken1, degisken2 = degisken2, degisken1

Fonksiyon ve Metot

```
myList = [10, 1, 8, 3, 5]
```

```
myList[0], myList[4] = myList[4], myList[0]
```

```
myList[1], myList[3] = myList[3], myList[1]
```

```
print(myList)
```

- Peki ya liste 100 eleman içeriyorsa ?

Fonksiyon ve Metot

```
myList = [10, 1, 8, 3, 5]
```

```
u = len(myList)
```

```
for i in range(u // 2):
```

```
    myList[i], myList[u - i - 1] = myList[u - i - 1], myList[i]
```

```
print(myList)
```

Fonksiyon ve Metot

- 3.1.4.13 LAB

5. Kısım

Sıralama

- Şimdi bir listedeki elemanların nasıl sıralanacağını öğrenme zamanı.
- Bir listenin iki şekilde sıralanabileceğini varsayalım:
 - artan (başka bir değişle - azalan olmayan) – komşu elemanların her çiftinde, önceki eleman sonraki büyük değildir;
 - azalan (başka bir değişle - artmayan) - komşu elemanların her çiftinde, önceki eleman sonraki daha küçük değildir.

Sıralama

- Bir listeyi artan düzende sıralayalım.

8	10	6	2	4
---	----	---	---	---

8	6	10	2	4
---	---	----	---	---

8	6	2	10	4
---	---	---	----	---

8	6	2	4	10
---	---	---	---	----

Sıralama

- Bir listeyi artan düzende sıralayalım.

6	8	2	4	10
6	2	8	4	10
6	2	4	8	10

Sıralama

- Bir listeyi artan düzende sıralayalım.

2	6	4	8	10
---	---	---	---	----

2	4	6	8	10
---	---	---	---	----

Sıralama

- Birinci ve ikinci elemanları alıp karşılaştırılır; eğer bu elemanlar yanlış sırada ise (yani; birincisi ikinciden daha büyük ise), bu elemanları değiştirilir; eğer sıralama doğru ise bir işlem yapılmaz.
- Aynı işlemi ikinci ve üçüncü elemanlar için yapılır.
- Daha sonra listede devam edip üçüncü ve dördüncü elemanlara bakılır.
- Listedeki elemanların hepsini geçtikten sonra, yeni listede ikinci bir tura başlanır.
- Bu sıralama yöntemine *bubble sort* adı verilir.

Sıralama

- Bu algoritmanın yapısı basittir: Bitişik elemanları karşılaştırır ve gerekenleri değiştirerek hedefimize ulaşırız.

<https://www.youtube.com/watch?v=lyZQPjUT5B4>

Sıralama

- Tüm listeyi sıralamak için kaç tura ihtiyacımız var?
- Başka bir değişken kullanarak bu sorunu çözebiliriz; Bu değişkenin görevi, geçiş sırasında herhangi bir yer değişikliği yapılmış olup olmadığını gözlemlemektir
- Bir değişiklik yoksa liste zaten sıralanmıştır ve başka tura ihtiyaç yoktur.
- *degistirildi_mi* adında bir değişken oluşturalım ve değişme işlemi olmadığını belirtmek için bu değişkene *False* değerini atayalım. Eğer bir değişme olursa *degistirildi_mi* değişkenini *True* olarak güncelleriz.

Sıralama

```
benimListem = [8, 10, 6, 2, 4]  
degistirildi_mi = True
```

```
while degistirildi_mi:  
    degistirildi_mi = False                                # Henüz bir değişiklik olmadı  
    for i in range(len(benimListem) - 1):  
        if benimListem[i] > benimListem[i + 1]:  
            degistirildi_mi = True                          # Değişiklik yapıldı!  
            benimListem [i], benimListem [i + 1] = benimListem [i + 1], benimListem [i]  
  
print(benimListem)
```

Sıralama

- Bununla beraber Python'ın kendi sıralama mekanizmaları da vardır.
- Yeterli sayıda kullanıma hazır kodlar olduğu için kimsenin kendi sıralama mekanizmasını yazması gerekmez.

```
benimListem = [8, 10, 6, 2, 4]
```

```
benimListem.sort()
```

```
print(benimListem)
```

- Tüm listeleri olabildiğince hızlı bir şekilde sıralayan `sort()` adında bir metot vardır.

6. Kısım

Liste Operasyonları

- Listeler, bellekte sıradan (skaler) değişkenlerden farklı şekilde saklanır.

```
liste1 = [1]  
liste2 = liste1  
liste1[0] = 2  
print(liste2)
```

Liste Operasyonları

- Sıradan bir değişkenin ismi, değişkenin içeriğin ismidir;
- Bir listenin ismi ise listenin saklandığı hafıza konumunun ismidir.
- `liste2 = liste1` şeklinde yapılan atama, dizinin adını kopyalar, içeriğini değil.
- Aslında, iki isim (`liste1` ve `liste2`), bilgisayar belleğindeki aynı yeri tanımlar.
- Bu listelerden herhangi birinde yapılan bir değişiklik diğerini de etkiler.

Liste Operasyonları

- Dilimleme, bir listenin veya listenin bir bölümünün yepyeni bir kopyasını almamızı sağlayan bir Python özelliğidir.
- Aşağıdaki şekilde listenin içeriğini kopyalanır.

```
liste1 = [1]
```

```
liste2 = liste1[:]
```

```
liste1[0] = 2
```

```
print(liste2)
```


Liste Operasyonları

- Kodun göze çarpmayan ve [:] olarak tanımlanan kısmı ile yepyeni bir liste oluşturabilir.
- Dilimlemenin en genel kullanım yapısı aşağıdaki gibidir:
benimListem[başlangıç:bitiş]
- Bu şekilde, kaynak listedeki başlangıç ile (bitiş-1) arasındaki indekslerin elemanları kullanılarak yeni bir (hedef) liste oluşturulur.
- Not: bitiş değil (bitiş-1). Bitiş değerine eşit olan indeksteki eleman, dilimleme sonucunda yer almayan ilk elemandır.

Liste Operasyonları

- Kodun göze çarpmayan ve `[:]` olarak tanımlanan kısmı ile yepyeni bir liste oluşturabilir.
- Dilimlemenin en genel kullanım yapısı aşağıdaki gibidir:
benimListem[başlangıç:bitiş]
- Bu şekilde, kaynak listedeki başlangıç ile (bitiş-1) arasındaki indekslerin elemanları kullanılarak yeni bir (hedef) liste oluşturulur.
- Not: bitiş değil (bitiş-1). Bitiş değerine eşit olan indeksteki eleman, dilimleme sonucunda yer almayan ilk elemandır.

Liste Operasyonları

- 3.1.6.7
- 3.1.6.8
- 3.1.6.9

7. Kısım

Liste Uygulamaları

- Bir listenin skaler değişkenlerden daha karmaşık yapıda elemanları olabilir. Örneğin:

```
ikinci_sıra = []  
for i in range(8):  
    ikinci_sıra.append("BEYAZ_PİYON")
```

- Bu kod parçası, adı `ikinci_sıra` olan bir liste oluşturur.
- Sonra 8 tane beyaz piyon `ikinci_sıra` ya eklenir.

Liste Uygulamaları

- Aynı sonucu liste oluştururken de elde edebiliriz.

ikinci_sira = ["beyaz piyon" for i in range(8)]

- Bu kod parçasında liste oluşturulurken, programın çalışması esnasında, elemanlar eklendi.
- Listeyi doldurmak için kullanılacak veri (*beyaz_piyon*)
- Verinin listeye kaç kez ekleneceğini belirten kısım (*for i range(8)*)

Liste Uygulamaları

- Satranç tahtasındaki boş yerleri belirtmek için **BOŞ** adlı önceden tanımlanmış bir sembol **varsayalım**.
- Tamamen boş bir satranç tahtası oluşturmak için **listelerin eleman olarak kullanıldığı bir liste tanımlayalım**.

```
satranc_tahtasi = []
```

```
for i in range(8):
```

```
    sira = [BOŞ for i in range(8)]
```

```
    satranc_tahtasi.append(sira)
```

Liste Uygulamaları

- Bu işlem sonucu satranç_tahtasi değişkeni iki boyutlu bir dizi haline geldi.
- Bu gibi diziler cebirde matris olarak isimlendirilir.
- Bu yapı alttaki gibi daha kısa şekilde de oluşturulabilir.

satranc_tahtasi = [[EMPTY for i in range(8)] for j in range(8)]

- İç taraftaki parantezin sırayı oluştururken, dış taraf sıraları içeren bir liste oluşturur.

Liste Uygulamaları

- Satranç tahtasında seçilmek istenen alana erişmek için iki indeks kullanılması gerekmektedir.
- İlk indeks sıranın seçilmesini sağlarken, ikinci indeks o sıranın içindeki alanın seçilmesini sağlar.
- Şimdi satranç tahtasına kaleleri ekleyelim.

satranc_tahtasi[0][0] = “kale”

satranc_tahtasi[0][7] = “kale”

satranc_tahtasi[7][0] = “kale”

satranc_tahtasi[7][7] = “kale”

Liste Uygulamaları

	A	B	C	D	E	F	G	H	
8	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]	[0][5]	[0][6]	[0][7]	8
7	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]	[1][5]	[1][6]	[1][7]	7
6	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]	[2][5]	[2][6]	[2][7]	6
5	[3][0]	[3][1]	[3][2]	[3][3]	[3][4]	[3][5]	[3][6]	[3][7]	5
4	[4][0]	[4][1]	[4][2]	[4][3]	[4][4]	[4][5]	[4][6]	[4][7]	4
3	[5][0]	[5][1]	[5][2]	[5][3]	[5][4]	[5][5]	[5][6]	[5][7]	3
2	[6][0]	[6][1]	[6][2]	[6][3]	[6][4]	[6][5]	[6][6]	[6][7]	2
1	[7][0]	[7][1]	[7][2]	[7][3]	[7][4]	[7][5]	[7][6]	[7][7]	1
	A	B	C	D	E	F	G	H	

Liste Uygulamaları

- Eğer C4’e at taşını eklemek istersek:

satranc_tahtasi[4][2] = “at”

- E5’e bir piyon koyalım:

satranc_tahtasi[3][4] = “piyon”

Liste Uygulamaları

- İki boyutlu bir listede herhangi bir elemanı bulmak için iki koordinata ihtiyacımız var.
 - Dikey koordinat (satır numarası)
 - Yatay koordinat(sütün numarası)
- Hava durumu verilerinin otomatik olarak alındığı bir yazılım geliştirdiğimizi düşünelim.
- Hava durumunu ileten cihaz saatlik olarak veriyi gönderiyor ve kayıtlar aylık olarak tutuluyor.

Liste Uygulamaları

- Toplam olarak $24 \times 31 = 744$ adet hava durumu verisi var.
- Şimdi bütün bu verilerin saklanabileceği bir liste tasarlayalım.
- Öncelikle, bu uygulama için hangi veri tipinin uygun olduğuna karar verelim. Kullanılan termometrenin sıcaklığı 0.1 duyarlılık ile ölçtüğü bilindiğinden bu uygulama için reel sayıları(float) kullanmak doğru olacaktır.

Liste Uygulamaları

- Daha sonra her saat için keyfi olarak verilmiş bir değer ile bütün saat listelerini oluşturalım.
- Bu durumda bir satırda 24 tane eleman bulunacaktır.
- Oluşturduğumuz 24 elemanlı liste bir günü temsil edeceğinden bu gün listelerinden 31 tane oluşturmamız gerekir.(Bir ayın 31 gün olduğunu varsayıyoruz)
- Bu şartları sağlayacak örnek (s saatleri, g günleri tanımlar):

sicakliklar = [[0.0 for s in range(24)] for g in range(31)]

Liste Uygulamaları

- Alınan veriler doğrultusunda aylık ortalama öğlen sıcaklığını hesaplayabiliriz.
- Saat 12 de alınmış 31 kaydı toplayıp 31'e bölümü bize ortalama sıcaklığı verecektir.
- Gece yarısı alınan verinin alınan ilk veri olduğu düşünülürse:

toplam = 0.0

for gun in sicakliklar:

toplam += gun[11]

ortalama = toplam / 31

print("Saat 12 için ortalama sıcaklık: ", ortalama)

Liste Uygulamaları

- Ayın en yüksek sıcaklık değerini bulalım.

```
en_yuksek = -100.00
```

```
for gun in sicakliklar:
```

```
    for gecici in gun:
```

```
        if gecici > en_yuksek:
```

```
            en_yuksek = gecici
```

```
print("En yüksek değer:", en_yuksek)
```


Liste Uygulamaları

- “*gun*” değişkeni, sıcaklıklar listesinin bütün elemanlarını teker teker tarar.
- “*gecici*” değişkeni, bir gün için oluşturulmuş sıcaklık değerleri listesini teker teker tarar.

Liste Uygulamaları

- Şimdi Saat 12’de hava sıcaklığının en az 20 derece olduğu günlerin sayısını bulalım. Bu günlere sıcak gün diyelim.

sicakGunSayisi = 0

for gün in sicakliklar:

if gün[11] > 20.0:

sicakGunSayisi += 1

print(sicakGunSayisi, " gün çok sıcaktı.")

Liste Uygulamaları

- Python'da iç içe liste yazma konusunda bir sınır yoktur. Şimdi üç boyutlu bir liste örneği yapalım.
- Bir otel düşünelim. Bu büyük otelin üç tane binası olsun. Her bina on beş katlı ve her katta yirmi oda olsun. Bizim projemiz her oda için kullanımda/uygun bilgisini bir dize(array) üzerinde toplamak.

Liste Uygulamaları

- İlk aşama: Dizedeki elemanların tipini belirlemek. Bu durum için Boole değerler (True/False) uygundur.
- İkinci aşama: İstenen yapının analizi.

Elimizdeki verilerin özeti: 3 bina,15 kat,20 oda

odalar = [[[False for oda in range(20)] for kat in range(15)] for bina in range(3)]

Liste Uygulamaları

odalar = [[[False for oda in range(20)] for kat in range(15)] for bina in range(3)]

- İlk indeks (0'dan 2 ye kadar) bir bina seçmeyi sağlar.
- İkinci indeks(0'dan 14'e kadar) bir kat seçmeyi sağlar.
- Üçüncü indeks(0'dan 19'a kadar)bir oda seçmeyi sağlar.

Liste Uygulamaları

- Şimdi yeni evli bir çift için otelde yer ayırtalım.
- İstenen oda ikinci binanın onuncu katında 14 numaralı oda.

odalar[1][9][13] = True

- Birinci binanın beşinci katındaki,dolu olan, 2 numaralı odayı boşaltalım.

odalar[0][4][1] = False

Liste Uygulamaları

- Üçüncü binada 15. Katta kaç boş oda olduğunu öğrenelim.

```
bos_oda_sayisi = 0
```

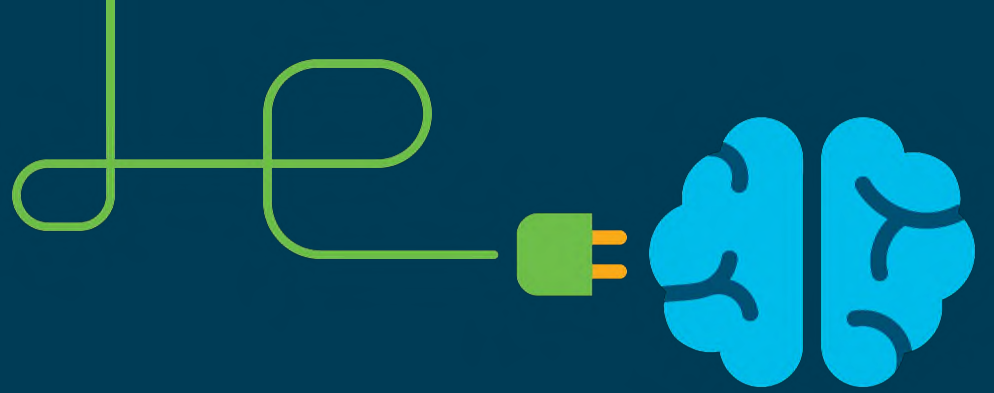
```
for oda_numarasi in range(20):
```

```
    if not odalar[2][14][ oda_numarasi]:
```

```
        bos_oda_sayisi += 1
```

```
print(bos_oda_sayisi)
```






MEB OYGM & Cisco işbirliği ile,

Dijital Transformasyon Python Programlama Eğitimleri

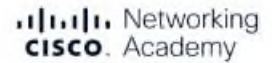
Kasım 2019



Enter the world of



python™



PCAP Programming Essentials in Python

Modul 4

1. Kısım

Fonksiyonlar

- Bazı kod parçalarını birden çok kez yazmak gerekebilir.
- Bu kod parçasını birden çok projede kullanmak gerekebilir.
- Bu durum, tekrarlanmış kod parçasını, yalıtılmış bir formda bir fonksiyon halinde yazma gereksinimi doğurur.
- Böylece kodun içinden tekrarlanmış parça çıkarılabilir ve yerinde aynı işi yapması için oluşturulan fonksiyon yazılabilir.

Fonksiyonlar

- Geliştirici, kodu(daha doğrusu: problemi) parçalara bölüp, iyi izole edilmiş parçalar haline getirir.
- Bütün bu parçaları birer fonksiyon haline getirir. Bu fonksiyonları gerektiğinde kullanmaya başlar.
- Bu şekilde yapılandırılmış kod, uygulamanın üzerinde çalışmayı bir hayli kolaylaştırır.
- Çünkü kodun parçaları ayrı ayrı eklenmiş ve tek tek test edilmiştir.
- Bu işleme çoğunlukla ayrıştırma(dekompozisyon) denir.

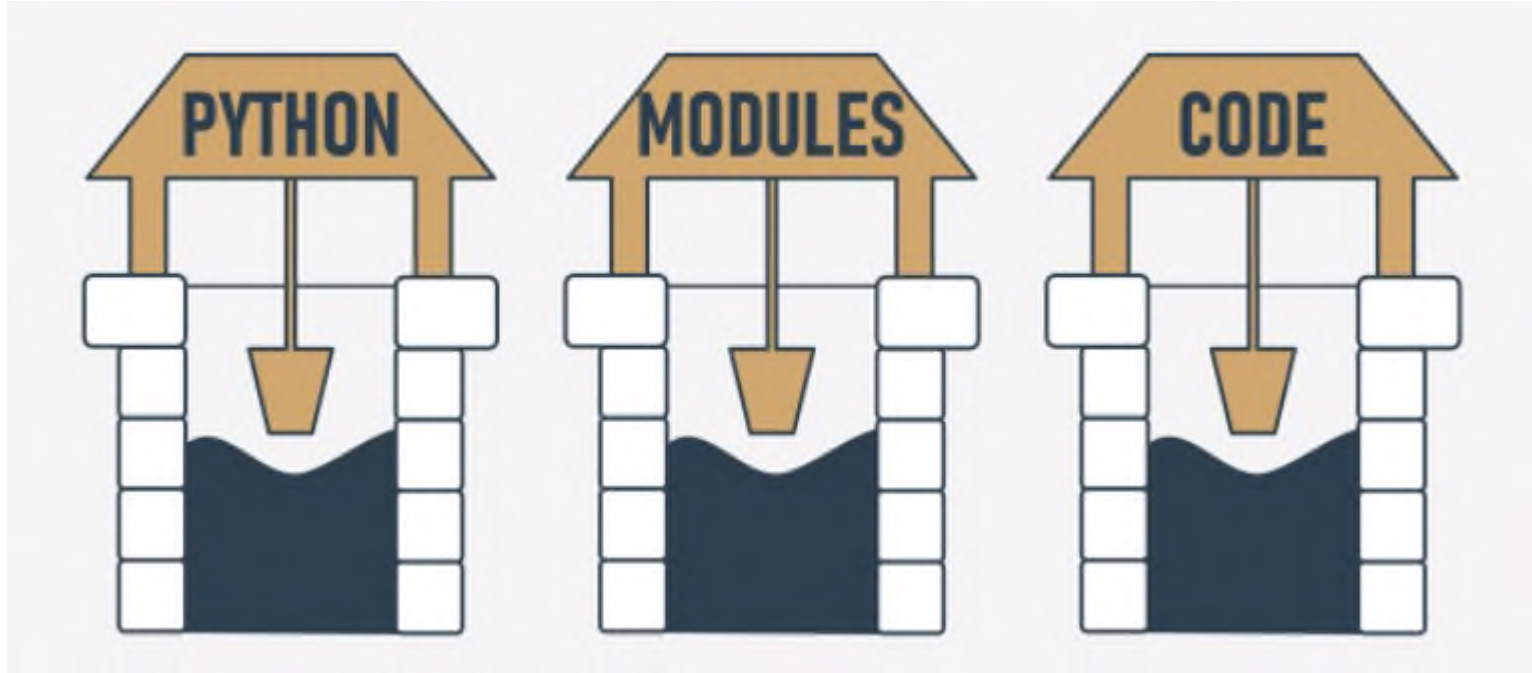
Fonksiyonlar

- Üzerinde çalışılacak problem bazı durumlarda çok büyük ve karmaşık olabilir ve bu gibi durumlarda çoğunlukla bir grup geliştirici birlikte çalışırlar.
- Problem, her geliştiricinin verimli çalışabilmesi ve iş bölümü yapılabilmesi için parçalara bölünür.
- Bu tür bir ayrıştırmanın amacı sadece işi parçalara ayırmak değil ayrıca sorumluluğu da birçok geliştirici üzerine dağıtmaktır.

Fonksiyonlar

- Tüm geliştiriciler anlaşılır şekilde tanımlanmış ve açıklanmış fonksiyonlar hazırlar.
- Sonra bu fonksiyonlar birleştirilerek, kullanıma hazır bir modülü (daha sonra bahsedilecek) oluşturur. Bu modüller ile son ürün ortaya çıkar.
- Eğer yapılacak iş birden çok geliştiriciye ayrıştırılmış problemler olarak verilirse, proje birbirinden ayrı olarak hazırlanmış fonksiyonlardan oluşan farklı modüller ile hayata geçirilmiş olur.

Fonksiyonlar



Fonksiyonlar

- Alttaki kodu inceleyelim.
- Burada amacımız tekrarlanmış kod parçalarını fonksiyona çevrilebileceği bir örnek sunmaktır.

```
print("Bir değer girin: ")
```

```
a = int(input())
```

```
print("Bir değer girin: ")
```

```
b = int(input())
```

```
print("Bir değer girin: ")
```

```
c = int(input())
```

Fonksiyonlar

- Bu örnekte print() fonksiyonu aynı mesaj ile çağırılmıştır.
- Eğer konsola yazılan mesaja biraz daha nazikçe (örn: Lütfen ile) başlamak isterseniz, nasıl bir işlem yapabilirsiniz?

```
print("Bir değer girin: ")
```

```
a = int(input())
```

```
print("Bir değer girin: ")
```

```
b = int(input())
```

```
print("Bir değer girin: ")
```

```
c = int(input())
```

Fonksiyonlar

- Bu durum karşısında, bir süre harcayarak kodunuzun içinde geçen tüm mesajları tek tek değiştirmeniz gerekecektir.
- Tekrarlanan kodu ayırmanın bir yolu var mıdır?
- Kodu tekrar kullanılabilir yapabilir miyiz?
- Bir noktadan yapılan değişiklik, kodun kullanıldığı her yere etki edebilir mi?
- Bu soruların cevabı “evet”tir.
- Fonksiyonlar tam olarak bunun için kullanılır.

Fonksiyonlar

- Böyle bir fonksiyonu nasıl oluşturabiliriz?
- Önce tanımlamamız gerekiyor. Burada tanımlama (define) kelimesi önemli.
- En basit şekilde bir fonksiyon tanımlama işlemi şu şekilde yapılır:

```
def fonksiyonİsmi():  
    fonsiyonİçeriği
```

Fonksiyonlar

- Bir fonksiyon her zaman *def* (define) anahtar kelimesiyle tanımlanır.
- *def* anahtar kelimesinden sonra fonksiyonun ismiyle devam edilir. (Fonksiyon isimlendirmedeki kurallar değişken isimlendirmedeki kurallar ile aynıdır.)

def fonksiyonİsmi():
fonksiyonİçeriği

Fonksiyonlar

- Fonksiyon ismi yazıldıktan sonra, bir çift parantez konur. (Bu örnekte parantezlerin içi boş olarak görünmekte, ama bu birazdan değişecek)
- Satır iki nokta üst üste işaretiyle bitirilir.

```
def fonksiyonİsmi():  
    fonsiyonİçeriği
```


Fonksiyonlar

- *def* ile başlayan satırdan bir sonraki satır ile başlayan kısma o fonksiyonun içeriği/gövdesi denir.
- Fonksiyon içeriğine indentation ile başlanır.
- Bir fonksiyonun en az bir satırlık içeriği olmalıdır. Bu içerik fonksiyon her çağrıldığında çalışır.
- Not: Fonksiyon iç içe yapının bittiği yerde biter. Buna dikkat etmek gerekir.

Fonksiyonlar

- Şimdi bir fonksiyon tanımlayalım.

def mesaj():

print("Bir değer girin: ")

- Bu fonksiyonu kodumuza ekleyelim:

def mesaj():

print("Bir değer girin: ")

print("Biz burada başlıyoruz.")

print("Burada bitiriyoruz.")

Fonksiyonlar

- Fonksiyonu henüz hiç kullanmadık.
- Kodun hiçbir satırında fonksiyon çağrılmadı.
- Yani Python bu fonksiyon tanımını okudu ve hafızasına kaydetti, fakat çalıştırmadı.
- Şimdi başlatma ve bitirme mesajlarının arasında fonksiyonumuzu çağıralım.

Fonksiyonlar

```
def mesaj():  
    print("Bir değer girin: ")  
  
print("Biz burada başlıyoruz.")  
mesaj() #fonksiyon çağrısı  
print("Burada bitiriyoruz.")
```

Fonksiyonlar



The diagram illustrates the flow of execution for a function. A red arrow points from the title 'Fonksiyonlar' to the function definition. Another red arrow points from the function call 'mesaj()' to the function definition. A third red arrow points from the function definition to the function call. A fourth red arrow points from the function call to the end of the code block. The code block is enclosed in a black L-shaped border.

```
def mesaj():  
    print("Bir değer girin: ")
```

```
print("Biz burada başlıyoruz.")  
mesaj() #fonksiyon çağırısı  
print("Burada bitiriyoruz.")
```

Fonksiyonlar

- Bir fonksiyon çalıştırıldığında, Python fonksiyonun yerini hatırlar ve o fonksiyonun içerisine girer.
- Fonksiyonun içeriği/gövdesi çalıştırılır.
- Fonksiyon en sona geldiğinde Python'ı fonksiyonun çalıştırıldığı yere dönmeye zorlar.
- Henüz tanımlanmamış bir fonksiyon çağırılmamalıdır.

Fonksiyonlar

- Python kodu yukarıdan aşağıya doğru okur. Fonksiyon çağırma işleminde eğer fonksiyon tanımlanmamış ise daha alt satırlarda bu fonksiyon aranmaz. Bu nedenle fonksiyon tanımını doğru yere (çağrılmadan önce) konmalıdır.
- Fonksiyonun ismi ile herhangi bir değişkenin ismi aynı olmamalıdır.
- Örneğin bir önceki örnekte “mesaj” ismi için bir değer atanması durumunda Python fonksiyon tanımını unuttur. Fonksiyonun kullanılabilirliğini kaybeder.

2. Kısım

Fonksiyon Parametreleri

- Fonksiyonlar çağrılırken bir ya da daha fazla sayıda değişken alıp, bu değişken üzerinde istenilen işlemleri gerçekleştirebilirler.
- Bu değişkenlere fonksiyon parametreleri ismi verilir.

```
def fonksiyon(parametre):  
    ### fonksiyon gövdesi
```

Fonksiyon Parametreleri

- Bir parametreyi diğer değişkenlerden farklı ve özel olmasını sağlayan faktörler:
 - 1-) Parametreler sadece tanımlandıkları fonksiyonun içerisinde var olurlar.
 - 2-) Bir parametre sadece *def* ile başlayan satırdaki parantezlerin içerisinde tanımlanabilir.
 - 3-) Fonksiyon çağrılmak istendiğinde, parantezler arasında yazılan argümanlar ile parametrelere değer atanır.

Fonksiyon Parametreleri

- Parametreler sadece fonksiyon içerisinde var olurlar, argümanlar ise fonksiyonun dışında bulunurlar.
- Argümanlar istenen parametreye değer atayan taşıyıcılardır.
- Şimdi “mesaj” fonksiyonunu yeniden düzenleyelim. Önce *def* kısmı ile başlayalım.

```
def mesaj(sayi):  
    ###
```

Fonksiyon Parametreleri

- Fonksiyonumuz, içerisine *sayı* adında tek bir parametre olarak tanımlanmıştır.
- Bir parametre için sıradan bir değişken kullanılabilir.
- Parametre başka hiçbir yerden görülmez.
- Şimdi fonksiyonun gövdesi/içeriği ile devam edelim.

def mesaj(sayı):

print("Bir sayı giriniz:", sayı)

Fonksiyon Parametreleri

- Bir parametre kullandık fakat bu parametreye henüz bir değer atamadık.
- Bir fonksiyon tanımında bir veya daha fazla parametre kullanılmışsa, fonksiyon çağrıldığında da aynı sayıda parameter kullanılması gerekiyor.

```
def mesaj(sayı):  
    print("Bir sayı giriniz:", sayı)
```

Fonksiyon Parametreleri

- Kod içerisindeki değişken isimlerinin, fonksiyon tanımındaki parametre isimleri ile aynı olması bir sorun oluşturmaz.

```
def mesaj(sayi):  
    print("Bir sayı giriniz::", sayi)
```

```
sayi = 1234  
mesaj(1)  
print(sayi)
```

Fonksiyon Parametreleri

- Bu duruma gölgeleme (shadowing) adı verilir.
- X (*sayi*) parametresi aynı isimli değişkeni gölgede bırakır.
- Ama gölgeleme sadece fonksiyonun içeriğini bulunduğu bölgede geçerlidir.
- “*sayi*” *parametresi* kodun içerisinde tanımlanan “*sayi*” *değişkeninden* tamamen farklıdır.

Fonksiyon Parametreleri

- Bir fonksiyona istediğiniz kadar parametre ekleyebilirsiniz fakat ne kadar parametre koyarsanız, bu parametrelerin rol ve amaçlarını ezberlemek o kadar zorlaşır.
- Şimdi fonksiyonumuzu biraz değiştirelim. Bir parametre daha ekleyelim.
- Bu aynı zamanda fonksiyonu çağırırken de iki ayrı argüman vermemizi gerektirir.

def mesaj(tip,sayi):

print("Lütfen bir", tip, "giriniz", sayi)

Fonksiyon Parametreleri

- Fonksiyon çağrılarında belirtilen argümanların yazılma sırasının, fonksiyon tanımında kullanılan parametrelerin sırası ile aynı olması durumuna *pozisyonu ile parametre atama* denir.
 - Bu yöntemde 1. sırada yazılan parametre, 1. sırada yazılan argüman ile çağrılır. 2. sıradaki parameter, 2.sıradaki argüman ile şeklinde devam eder.
 - Günlük hayatta da çoğu sosyal etkileşimde farkında olmadan, pozisyonu ile parametre atama işlemini kullanır.
- Örn: İsim-Soyisim (Tabi James Bond değilseniz 😊)

Fonksiyon Parametreleri

- Python parametre atama işlemi için ikinci bir yöntem de sunar.
- Bu yöntem ile argüman, parametrenin ismiyle atanır, pozisyonuyla değil. Bu yönteme *anahtar kelime ile parametre atama* denir.

```
def kendini_tanıt(ismin, soyismin):
```

```
    print("Merhaba, Ben ", ismin, soyismin)
```

```
kendini_tanıt(ismin = "James", soyismin = "Bond")
```

```
kendini_tanıt(soyismin = "Arkın", ismin = "Cüneyd")
```

Fonksiyon Parametreleri

- Burada işleyiş, parametrelere atanmak istenen değerden önce parametre ismini ve "=" sembolünü sırayla yazmaktır.
- Pozisyon önemli değildir. Her argümanın hangi parametreye atanacağı fonksiyon çağırılırken parametre isimleri kullanılarak sabitlenmiştir.
- Var olmayan bir parametre ismine bir değer atayamazsınız.

(soyadın = “Arkın“, adın = “Cüneyd”)

Fonksiyon Parametreleri

- İki tip parametre atama birlikte kullanılabilir.
- Burada dikkat edilmesi gereken tek bir kural var: pozisyona bağlı argümanlar, anahtar kelimeye bağlı argümanlardan sonra yazılmaz. Örneğin aşağıdaki fonksiyonu ele alalım.

```
def topla(a, b, c):
```

```
    print(a, "+", b, "+", c, "=", a + b + c)
```

- Bu fonksiyonun görevi argümanlarını toplamak ve göstermektir.

Fonksiyon Parametreleri

- Fonkiyonu *topla(1,2,3)* şeklinde çağırabiliriz, çıktısı $1 + 2 + 3 = 6$ olur.
- Bu çağırma şeklini değiştirerek tamamen anahtar kelime ile atamayı kullanalım: *topla(c = 1, a = 2, b = 3)*, bu kodun çıktısı $2 + 3 + 1 = 6$ olacaktır.
- Simdi her iki tipi beraber kullanalım:
topla(3, c = 1, b = 2),
- Bu kodun çıktısı $3 + 2 + 1 = 6$ olacaktır.

Fonksiyon Parametreleri

topla(3, c = 1, b = 2),

- Argüman (3), pozisyonu ile parametre atama yöntemi kullanarak a parametresine atanacaktır.
- c ve b anahtar kelimeleri kullanarak atandıklarının için pozisyonlarının önemi yoktur.

Fonksiyon Parametreleri

- Şimdi farklı bir çağrıma yapalım.
a parametresini iki yöntem ile atayalım: *topla(3, a = 1, b = 2)*

Python yanıtı: *TypeError: topla() got multiple values for argument 'a'*

Anlamı: *TipHatası: topla() a argümanı için birden çok değer aldı.*

Fonksiyon Parametreleri

- Bazı durumlarda belirli parametrelere atanan değerler çoğunlukla aynı/sabit olabilir.
- Bu argümanlara varsayılan(önceden belirlenmiş) bir değer verilebilir. Eğer çağırma sırasında parametre atanmaz ise varsayılan değer atanır.
- Türkiye’de en çok kullanılan soyismin Yılmaz olduğunu varsayalım. Bu bilgi ile kendini_tanıt fonksiyonumuzu güncelleyelim.

Fonksiyon Parametreleri

- Varsayılan değer atamak için parametre tanımını yaparken isimden sonra = sembolünü ardından varsayılan olarak kullanılacak değeri girmeniz yeterlidir.

```
def kendini_tanıt(ismin, soy_ismin="Yılmaz"):  
    print("Merhaba, Ben ", ismin, soy_ismin)
```

```
kendini_tanıt("Ahmet", "Ünsal")  
kendini_tanıt("Hayriye")
```

3. Kısım

Fonksiyon Parametreleri

- Daha önce kullandığımız fonksiyonların tamamının bir takım etkileri vardı. Bir metin oluşturup konsola gönderiyorlardı.
- Fonksiyonların, matematikte olduğu gibi sonuçları da olabilir.
- Bir fonksiyonun sonuç değeri üretebilmesi için *return* talimatı kullanılır.

Fonksiyon Parametreleri

- “*return*” talimantının iki farklı seçeneği vardır. Bunları ayrı ayrı inceleyelim:
- İlki, sadece *return* anahtar kelimesinden oluşur. Devamında hiçbir şey yazılmaz.
- Fonksiyon içinde kullanıldığında ani bir tepki ile fonksiyon yürütülmesi bitirilir ve fonksiyonun çalıştırıldığı kod satırına geri dönülür.

Fonksiyon Parametreleri

- Eğer fonksiyon herhangi bir sonuç üretmeyecekse, *return* talimatını kullanmaya gerek kalmaz.
- Şimdi *return* etkisini birlikte inceleyelim.
- Yazacağımız fonksiyonu herhangi bir argüman vermeden çağırmamız durumunda, Fonksiyon "Mutlu Yıllar!" yazacaktır.
- Eğer fonksiyon argüman olarak *False* değerini alırsa, fonksiyonun çalışma şekli değişecektir.
- Konsola "Mutlu Yıllar!" yazmadan *return* etkisiyle fonksiyon bitecektir.

Fonksiyon Parametreleri

```
def mutluYillar(mutlu_yillar_dileyeyim = True):  
    print("Üç...")  
    print("İki...")  
    print("Bir...")  
    if not mutlu_yillar_dileyeyim:  
        return  
    print("Mutlu Yıllar!")
```

Fonksiyon Parametreleri

- İkinci yöntemde, `return` anahtar kelimesinin ardından bir ifade gelebilir. Örneğin:

fonksiyon():

return ifade

- Bu şekilde *return* talimatının kullanılması iki sonuç doğurur:

1-) Fonksiyon içeriğinde kullanıldığında ani bir tepki ile fonksiyon yürütülmesi bitirilir. (İlk kullanım yapısıyla aynı)

2-) Dahası, fonksiyon *return* talimatının ardından yazılan ifadeyi çalıştırır, **değerini çağırıldığı satırda geri döner.**

Fonksiyon Parametreleri

```
def sıkıcı_bir_fonksiyon():  
    return 123  
x = sıkıcı_bir_fonksiyon ()  
print("sıkıcı_bir_fonksiyon'dan gelen cevap:", x)
```

- Kod parçacığı alttaki metni konsola yazar.
“*sıkıcı_bir_fonksiyon'dan gelen cevap: 123*”
- “*return*” talimatı bir ifade ile çalıştığında(örnekteki ifade de görüldüğü gibi), ifadenin değerini fonksiyonun çağrıldığı yere gönderir.

None

- “*None*” bir anahtar kelimedir ve anlamlı herhangi bir veri barındırmaz.
- Bir değişkene “*None*” değeri atanabilir. (veya bir fonksiyonun sonucu olarak kullanılabilir.)
- Bir değişkenin verisi hakkında bilgi almak için “*None*” değeri ile kıyaslanabilir.

None

- Bir fonksiyon sonuç olarak herhangi bir değer dönmez ise yani return ifadesi kullanılmaz ise Python geri dönecek değer *None* olacağını varsayar.
- Örneğin alttaki kodu inceleyelim.

```
def ilgincFonksiyon(n):
```

```
    if(n % 2 == 0):
```

```
        return True
```

None

```
def ilgincFonksiyon(n):  
    if(n % 2 == 0):  
        return True
```

- Bu fonksiyonu kontrol etmek için alttaki kod parçasını kullanalım.

```
print(ilgincFonksiyon(2))  
print(ilgincFonksiyon(1))
```

None

```
def ilgincFonksiyon(n):  
    if(n % 2 == 0):  
        return True  
print(ilgincFonksiyon(2)) # True  
print(ilgincFonksiyon(1)) # None
```

- Bir fonksiyonun çıktısı olarak *None* dönerse şaşırmayalım.
- Bazen bu çıktı, fonksiyonda yazılmış göze çarpmayan bir hata belirtisi de olabilir.

Fonksiyonlar

- Fonksiyonlara argüman olarak Python'da tanımlı bütün varlıklar verilebilir.
- Fakat o fonksiyonun aldığı argümanı kullanabileceğinden emin olunmalıdır.
- Bu duruma örnek olarak:

```
def liste_toplayıcı(liste):  
    toplam = 0  
    for eleman in liste:  
        toplam += eleman  
    return toplam
```

Fonksiyonlar

- Şimdi kodu alttaki gibi çalıştıralım.

```
def liste_toplayıcı(liste):  
    toplam = 0  
    for eleman in liste:  
        toplam += eleman  
    return toplam  
print(liste_toplayıcı([5, 4, 3]))
```

Fonksiyonlar

- Fonksiyonların döndüreceği değer(return) olarak Python'da tanımlı bütün varlıklar kullanılabilir.
- Örneğin:

```
def liste_olustur(n):  
    listem = []  
    for i in range(0, n):  
        listem.insert(0, i)  
    return listem  
print(liste_olustur(5))
```

4. Kısım

Fonksiyonlarda Kapsam

- Bir değişkenin isminin kapsamı kavramı, bu değişkenin doğru olarak tanınabildiği kod parçası anlamına gelir.
- Bir fonksiyondaki parametrenin kapsamı, o fonksiyonun gövdesidir.
- Fonksiyon içerisinde kullanılan parametre fonksiyonun dışından erişilemez.

Fonksiyonlarda Kapsam

- Bunu bir örnekle inceleyecek olursak.

```
def kapsamTest():  
    x = 123
```

```
kapsamTest()  
print(x)
```

- Bu kod hata verecektir.

NameError: name 'x' is not defined

Fonksiyonlarda Kapsam

- Başka bir örnekle, bir fonksiyonun dışında oluşturulmuş bir değişkenin o fonksiyonun içerisinden erişilip erişilemediğini kontrol edelim.

```
def benimFonksiyonum():  
    print("Acaba bu değişkeni tanıyor muyum? ", var)
```

```
var = 1  
benimFonksiyonum()  
print(var)
```

Fonksiyonlarda Kapsam

- Bu şu anlama gelmektedir:
Fonksiyonun dışında oluşturulmuş bir değişkenin kapsamına fonksiyonun içi de dahildir.
- Bu durum, aşağıdaki gibi, bazı istisnalar haricinde geçerlidir.

Fonksiyonlarda Kapsam

```
def benimFonksiyonum():  
    var = 2  
    print("Acaba bu değişkeni tanıyor muyum? ", var)
```

```
var = 1  
benimFonksiyonum()  
print(var)
```

Fonksiyonlarda Kapsam

- Fonksiyonun içerisinde oluşturulan “var” değişkeni, dışında oluşturulan değişken ile aynı değildir.
- Aynı isimle oluşturulmuş iki değişken vardır. Değişkenlerin kapsamına dikkat edelim!
- Fonksiyonun içeriğindeki değişken aynı isimle dışarıda oluşturulmuş değişkeni gölgelemektedir. (shadowing)
- Öyleyse kapsam kuralını daha doğru ifade etmek gerekir.

Fonksiyonlarda Kapsam

- Fonksiyonun dışında oluşturulmuş bir değişkenin kapsamına, fonksiyonun içerisinde aynı isimle tanımlanmış bir değişken daha olması durumu haricinde, fonksiyonun içi de dahildir.
- Bu aynı zamanda fonksiyonun dışında oluşturulmuş bir değişkenin kapsamının, sadece okuma (read) işlemi yapılması durumunda fonksiyonun içini kapsadığını gösterir.

Fonksiyonlarda Kapsam - global

- Peki bir fonksiyon, içerisinde tanımlanmamış bir değişkenin değerini değiştirebilir mi?
- Kısaca cevap evet değiştirebilir.
- Bu işlemi Python içerisinde tanımlı özel bir metod olan “global” ile yapabiliriz.
- Fonksiyonun içerisinde “global” ile oluşturulmuş bir değişkenin kapsamı fonksiyonun dışını da ulaşır. Global bir değişken olur.

Fonksiyonlarda Kapsam - global

- “global” metodu, Python’ı fonksiyon içerisinde yeni bir değişken oluşturmaktansa dışarıda oluşturulmuş değişken üzerinde işlem yapmaya zorlar.
- Şimdi bir başka örnek üzerinde fonksiyonların argümanları ile nasıl etkileşime girdiğini bir başka örnek üzerinde pekiştirelim.

Fonksiyonlarda Kapsam

```
def benimFonksiyonum(n):  
    print("Bende bir ", n, " değeri vardı.")  
    n += 1  
    print("Şimdi ", n, " oldu.")
```

```
var = 1  
benimFonksiyonum(var)  
print(var)
```

Fonksiyonlarda Kapsam

- Bu örnekte de görüleceği üzere, fonksiyon içerisinde bir parametrenin değerini değiştirmek fonksiyonun dışına etki etmez.
- Bu aynı zamanda fonksiyonun argümanın değerini kullandığını, argümanı kullanmadığını gösterir. Bu durum bütün skalerler için geçerlidir.
- “global” metodunu unutmayalım.

Fonksiyonlarda Kapsam

- Fonksiyonların bu özelliği liste tipinde değişkenler için de geçerli midir? Hep birlikte inceleyelim.

```
def benimFonksiyonum(liste1):  
    print(liste1)  
    liste1 = [0, 1]
```

```
liste2 = [2, 3]  
benimFonksiyonum(liste2)  
print(liste2)
```

Fonksiyonlarda Kapsam

- Kod çıktısına baktığımızda önceki kural ile uyumlu bir sonuç görürüz.
- Fonksiyonun içerisinde yapılan bir değişiklik dışına etki etmemiştir.
- Fakat listeler söz konusu olduğunda bu durum hep böyle olmayabilir.

Fonksiyonlarda Kapsam

- Aşağıdaki örneği inceleyelim.

```
def benimFonksiyonum(liste1):  
    print(liste1)  
    del liste1[0]
```

```
liste2 = [2, 3]  
benimFonksiyonum(liste2)  
print(liste2)
```

Fonksiyonlarda Kapsam

- “liste1” parametresinin değerini değiştirmedik. Fakat listeyi değiştirdik.
- Eğer argüman bir liste ise:
 - Listeye ait bir parametrenin değerini değiştirmek listede herhangi bir değişikliğe sebep olmaz.
 - Listeyi değiştirdiğimizde ise(parametreyi değil listeyi) bu değişiklik listeye yansıtılacaktır.

5. Kısım

Fonksiyonlarda Kapsam

- Şimdi değinilen bütün bu özellikleri örnekler üzerinde inceleyelim.
- Vücut kitle endeksini(BMI) hesaplayan bir fonksiyon yazalım.

$$\text{BMI} = \frac{\text{(weight in kilograms)} \text{ kg}}{\text{height in meters}^2 \text{ m}}$$

Fonksiyonlarda Kapsam

- Vücut Kitle Endeksini veren formül iki değer almaktadır:
 - kişinin ağırlığı (kg cinsinden)
 - kişinin boyu (metre cinsinden)

$$\text{BMI} = \frac{\text{(weight in kilograms)} \text{ kg}}{\text{height in meters}^2 \text{ m}}$$

Fonksiyonlarda Kapsam

- Vücut Kitle endeksi bir kişinin ağırlığının, boyunun karesine bölünmesi ile elde edilir.
- Öyleyse fonksiyonumuzu oluşturalım. İsmi bmi olsun.

```
def bmi(kilo, boy):  
    return kilo / boy ** 2
```

```
print(bmi(52.5, 1.65))
```

Fonksiyonlarda Kapsam

- Şimdi fonksiyona verilen değerlerin doğruluğunu test edecek bir kaç ekleme yapalım. .

```
def bmi(kilo, boy):  
    if boy < 1 or boy > 3 or \  
        kilo < 20 or kilo > 200:  
        return None  
    return kilo / boy ** 2
```

```
print(bmi(252.5, 1.65))
```

Fonksiyonlarda Kapsam

- “\” ifadesinin kullanımına dikkat edelim.

```
def bmi(kilo, boy):  
    if boy < 1 or boy > 3 or \  
    kilo < 20 or kilo > 200:  
        return None  
    return kilo / boy ** 2
```

```
print(bmi(252.5, 1.65))
```

Fonksiyonlarda Kapsam

- Şimdi kodumuza, emperyal sistemi kullananlar için, emperyal sistemi metrik sisteme dönüştüren fonksiyonlar ekleyelim.
- 1 libre (lb), 0.45359237 kilograma eşittir.

```
def lb_to_kg(lb):  
    return lb * 0.45359237
```

Fonksiyonlarda Kapsam

- 1 feet (ft), 0.3048 metreye ve 1 inç (in), 0.0254 metreye eşittir.

```
def ft_to_m(ft, inc):  
    return ft * 0.3048 + inc * 0.0254
```

Fonksiyonlarda Kapsam

- Bazen sadece feet değerini girip inç değerini girmek istemeyebilirsiniz. Bu nedenle inç değerine varsayılan bir değer (0.0) atayalım.

```
def ft_to_m(ft, inc=0.0):  
    return ft * 0.3048 + inc * 0.0254
```

- Şimdi bütün kodu bir araya getirelim ve bir örnek ile deneyelim.

Fonksiyonlarda Kapsam

```
def lb_to_kg(lb):  
    return lb * 0.45359237  
def ft_to_m(ft, inc=0.0):  
    return ft * 0.3048 + inc * 0.0254  
def bmi(kilo, boy):  
    if boy < 1 or boy > 3 or kilo < 20 or kilo > 200:  
        return None  
    return kilo / (boy ** 2)  
print (bmi(kilo = lb_to_kg(176), boy = ft_to_m(5, 7)))
```

Fonksiyonlarda Kapsam

- Şimdi kenar uzunluklarını kullanıcıdan alan ve verilen uzunluklara göre üçgenin doğruluğunu hesaplayan fonksiyonu yazalım.
- Bir üçgenin herhangi iki kenarının uzunlukları toplamı, üçüncü kenarın uzunluğundan büyük olmalıdır.
- Fonksiyon üç kenarın uzunluğunu parameter olarak alacak ve yukarıdaki kurala uyuyorsa True, uymuyorsa False dönecek.

Fonksiyonlarda Kapsam

```
def ucgenMidir(a, b, c):  
    if a + b <= c:  
        return False  
    if b + c <= a:  
        return False  
    if c + a <= b:  
        return False  
    return True  
print(ucgenMidir(1, 1, 1))  
print(ucgenMidir(1, 1, 3))
```

Fonksiyonlarda Kapsam

- Kodu biraz düzenlersek:

```
def ucgenMidir(a, b, c):  
    if  $a + b \leq c$  or  $b + c \leq a$  or  $c + a \leq b$ :  
        return False  
    return True  
print(ucgenMidir(1, 1, 1))  
print(ucgenMidir(1, 1, 3))
```

Fonksiyonlarda Kapsam

- Biraz daha düzenleyelim.

```
def ucgenMidir(a, b, c):  
    return a + b > c and b + c > a and c + a > b
```

```
print(ucgenMidir(1, 1, 1))  
print(ucgenMidir(1, 1, 3))
```

- Şimdi uzunlukları kullanıcıdan isteyelim.

Fonksiyonlarda Kapsam

```
def ucgenMidir(a, b, c):  
    return a + b > c and b + c > a and c + a > b  
a = float(input("İlk kenarın uzunluğunu giriniz: "))  
b = float(input("İkinci kenarın uzunluğunu giriniz: "))  
c = float(input("Üçüncü kenarın uzunluğunu giriniz: "))  
if ucgenMidir(a, b, c):  
    print("Tebrikler! Bu bir üçgendir.")  
else:  
    print("Malesef! Bu bir üçgen değildir.")
```

Fonksiyonlarda Kapsam

- Şimdi Heron formülü ile üçgenin alanine hesaplayalım.

$$s = \frac{a+b+c}{2}$$

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

Fonksiyonlarda Kapsam

def heron(a, b, c):

p = (a + b + c) / 2

*return (p * (p - a) * (p - b) * (p - c)) ** 0.5*

$$s = \frac{a+b+c}{2}$$

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

Fonksiyonlarda Kapsam

```
def heron(a, b, c):  
    p = (a + b + c) / 2  
    return (p * (p - a) * (p - b) * (p - c)) ** 0.5
```

```
def ucgeninAlanı(a, b, c):  
    if not ucgenMidir(a, b, c):  
        return None  
    return heron(a, b, c)
```

Fonksiyonlarda Kapsam

```
def heron(a, b, c):  
    p = (a + b + c) / 2  
    return (p * (p - a) * (p - b) * (p - c)) ** 0.5  
def ucgeninAlanı(a, b, c):  
    if not ucgenMidir(a, b, c):  
        return None  
    return heron(a, b, c)  
  
def ucgenMidir(a, b, c):  
    return a + b > c and b + c > a and c + a > b  
a = float(input("İlk kenarın uzunluğunu giriniz: "))  
b = float(input("İkinci kenarın uzunluğunu giriniz: "))  
c = float(input("Üçüncü kenarın uzunluğunu giriniz: "))  
if ucgenMidir(a, b, c):  
    print("Tebrikler! Bu bir üçgendir.")  
    print("Üçgenin alanı", ucgeninAlanı(a,b,c), "metre karedir.")  
else:  
    print("Malesef! Bu bir üçgen değildir.")
```

Fonksiyonlarda Kapsam

- Şimdi faktöriyel hesaplayan bir fonksiyon yapalım.

$$0! = 1$$

$$1! = 1$$

$$2! = 1*2$$

$$3! = 1*2*3$$

.

.

$$n! = 1*2*3*...*n-1*n$$

Fonksiyonlarda Kapsam

```
def faktoriyel(n):  
    if n < 0:  
        return None  
    if n < 2:  
        return 1  
  
    carpim = 1  
    for i in range(2, n + 1):  
        carpim *= i  
    return carpim  
  
for n in range(1, 6):  
    print(n, faktoriyel(n))
```

Fonksiyonlarda Kapsam

- Fibonacci seri açılımını hesaplayan bir fonksiyon yazalım.
- Şu şekilde hesaplanır:
 - Fibonacci 1, 1 dir.
 - Fibonacci 2, 1 dir.
 - Geri kalan bütün Fibonacci değerleri kendinden önceki iki sayının Fibonacci karşılığını toplayarak hesaplanır.

Fonksiyonlarda Kapsam

- Fibonacci seri açılımını hesaplayan bir fonksiyon yazalım.

```
fib1 = 1
fib2 = 1
fib3 = 1 + 1 = 2
fib4 = 1 + 2 = 3
fib5 = 2 + 3 = 5
fib6 = 3 + 5 = 8
fib7 = 5 + 8 = 13
```

Fonksiyonlarda Kapsam

```
def fib(n):  
    if n < 1:  
        return None  
    if n < 3:  
        return 1  
  
    elem1 = elem2 = 1  
    sum = 0  
    for i in range(3, n + 1):  
        sum = elem1 + elem2  
        elem1, elem2 = elem2, sum  
    return sum  
  
for n in range(1, 10):  
    print(n, "->", fib(n))
```

Tekrarlı Fonksiyonlar

- Tekrarlı (Rekürsif/Özyinelemeli) fonksiyon, kendi kendini çağıran fonksiyon demektir.
- Fibonacci serisi tanımını tekrarlı fonksiyon tanımına çok uymaktadır.

$$\text{Fib}(n) = \text{Fib}(n-1) + \text{Fib}(n-2)$$

- Faktöriyel hesaplamasında da tekrarlı fonksiyon özelliği kullanılabilir.

$$n! = (n-1)! * n$$

Tekrarlı Fonksiyonlar

```
def fib(n):  
    if n < 1:  
        return None  
    if n < 3:  
        return 1  
    return fib(n - 1) + fib(n - 2)
```

Tekrarlı Fonksiyonlar

```
def faktoriyel(n):  
    if n < 0:  
        return None  
    if n < 2:  
        return 1  
    return n * faktoriyel(n - 1)
```

6. Kısım

Tuple

- Python'da birden fazla veri tutabilen ve bu verilere sıra ile erişilmesine imkan veren veri tiplerine **sıralı dizi** tipinde veriler ismi verilir.
- Örneğin liste tipinde veriler sıralı dizi tipine örnek verilebilir.
- Sıralı dizi tipindeki veriler for döngüleri kullanımına çok uygundur.

Tuple

- Python'da kullanılan veri tiplerinin bir diğer özelliği değişebilir olmak ya da olmamaktır.
- Değişebilir tipte veriler, programın yürütülmesi esnasında güncellenebilme yeteneğine sahiptir. Bu tipte verilere yeni bir eleman atayabilir, çıkartabilir ve içeriğini değiştirebilirsiniz.
- Eğer bir veri değişebilir olmayan bir tipte veri ise bu işlemler yapılamaz.

Tuple

- Tuple(demet) tipinde veriler değişebilir olmayan verilere örnek verilebilir.
- Tuple tipinde verilerin liste tipindeki verilerden farkı değişebilir olmamalarıdır.
- Bir tuple tipinde veri aşağıdaki şekilde oluşturulabilir:

boşTuple = ()

Tuple

- Tuple oluşturmak için parantezlerin zorunlu olmasa da, parantez kullanımı tercih edilen yöntemdir.

birTuple = (1,2)

birTuple = 1,2

- Bir tuple tipinde verinin elemanlarını görüntülemek için liste tipi verilerde kullandığımız yöntemleri kullanabilirsiniz.

Tuple

```
birTuple = (1, 10, 100, 1000)
```

```
print(birTuple[0])
```

```
print(birTuple[-1])
```

```
print(birTuple[1:])
```

```
print(birTuple[:-2])
```

```
for elem in birTuple:
```

```
    print(elem)
```


Tuple

- Tuple tipinde verilen değişebilir olmadığını unutmayalım.
- Dolayısıyla tuple tipinde verilerin içeriğini değiştirmeye kalktığımızda Python hata verir.
- “*len()*” fonksiyonu ve “*in*” ve “*not in*” operatörleri tuple ile birlikte kullanılabilir.
- “*+*” operatörü iki tuple tipinde veriyi birbirine bağlayarak yeni bir tuple tipinde veri oluşturulabilir.
- “***” operatörü de tuple ile birlikte kullanılabilir.

Dictionary

- Python'da kullanılan bir diğer veri tipi de dictionary'dir.
- Dictionary(sözlük) tipinde veriler bir sıralı tip özelliği göstermezler.
- Dictionary tipinde veriler değişebilir özelliği olan verilerdir.
- Dictionary tipinde veriler key(anahtar) ve value(değer) çiftlerinden oluşur.

Dictionary

- Aynı isme sahip birden fazla key olmamalıdır.
- Bir key herhangi bir tipte olabilir.
- Dictionary'ler bir liste değildir.
- “*len()*” fonksiyonu dictionary tipinde veriler için de kullanılabilir.
- Dictionary tipinde veriler tek yönlü çalışır. Yani key değerleri ile arama yapabiliyorken value değerleri ile arama yapmak için tasarlanmamışlardır.

Dictionary

- Aşağıdaki yol ile dictionary tipinde bir veri oluşturulabilir.
dict = {key1: value1, key2: value2}
- Veri çiftleri süslü parantezler içerisine virgül ile ayrılarak yazılır.
- Key ve value çiftleri ise birbirlerinden iki nokta (:) ile ayrılır.
- Dictionary tipinde verilerin elemanları bellekte sıra ile (listelerde olduğu gibi) tutulmaz.

Dictionary

```
dict = {"cat" : "kedi", "dog" : "köpek", "horse" : "at"}  
telNumaraları = {'eşim' : 5551234567, 'ali' : 2565785431}  
alınacaklar = {1 : "ekmek", 2 : "süt"}  
boşDict = {}
```

```
print(dict["cat"])  
print(telNumaraları["ali"])  
print(alınacaklar[1])
```

Dictionary

- Olmayan bir key kullanılması durumunda Python hata verecektir.
- Bir key'in bir dictionary tipindeki very içerisinde var olup olmadığını anlamak için “*in*” ve “*not in*” operatörleri kullanılabilir.

Dictionary

```
dict = {"cat" : "kedi", "dog" : "köpek", "horse" : "at"}  
kelimeler = ['cat', 'lion', 'horse']
```

```
for kelime in kelimeler:  
    if kelime in dict:  
        print(kelime, "->", dict[kelime])  
    else:  
        print(kelime, "dict içerisinde yok")
```

Dictionary

- Dictionary tipindeki verilerde *for* döngüleri kullanılabilir mi?
- Cevap hem hayır hem evet.
- Hayır çünkü dictionary sıralı bir tip değil.
- Evet çünkü *for* döngüsüne uygun hale getirmememizi sağlayan özel araçlarımız var.

Dictionary

```
dict = {"cat" : "kedi", "horse" : "at", "dog" : "köpek"}
```

```
for key in dict.keys():  
    print(key, "->", dict[key])
```

```
for key in sorted(dict.keys()):  
    print(key, "->", dict[key])
```

Dictionary

- Dictionary tipinde veriler için kullanabileceğimiz bir diğer metod *item()* metodudur.
- *item()* metodu dictionary içerisindeki key-value çiftlerini bir tuple olarak döner.
- *values()* metodu da *keys()* metodu gibi çalışır fakat value üzerinde çalışır. *keys()* metodu key üzerinde çalışır.

Dictionary

```
dict = {"cat" : "kedi", "horse" : "at", "dog" : "köpek"}
```

```
for ingilizce, turkce in dict.items():  
    print(ingilizce, "->", turkce)
```

```
for turkce in dict.values():  
    print(turkce)
```

Dictionary

- Dictionary tipinde bir değişkene yeni bir veri eklemek çok kolaydır.

```
dict = {"cat" : "kedi", "horse" : "at", "dog" : "köpek"}
```

```
dict["duck"] = "ordek"
```

- Eğer var olan bir key'in value değerini değiştirmek istiyorsak aşağıdaki gibi bir işlem yapabiliriz.

```
dict["cat"] = "miyav"
```

Dictionary

- “*update()*” metodunu kullanarak ta yeni bir veri ekleyebiliriz.

dict.update({"duck" : "ordek"})

- “*del*” talimatı ile dictionary içerisindeki bir eleman silinebilir.

del dict['dog']

- *popitem()* metodu ile dictionary içerisinden rastgele bir eleman silinebilir.

Dictionary

- 4.1.6.9 Örnek

